TUGAS AKHIR – KI141502

RANCANG BANGUN KECERDASAN BUATAN ADAPTIF PADA PERMAINAN *TURN-BASED STRATEGY ADVANCE WARS*

MUHAMMAD ARIF ROHMAN HAKIM
NRP 5111100099

Dosen Pembimbing
Imam Kuswardayan, S.Kom., M.T.
Ridho Rahman Hariadi, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

[Halaman ini sengaja dikosongkan]

FINAL PROJECT - KI141502

DESIGN AND IMPLEMENTATION OF ADAPTIVE ARTIFICIAL INTELLIGENCE FOR TURN-BASED STRATEGY GAME ADVANCE WARS

MUHAMMAD ARIF ROHMAN HAKIM
NRP 5111100099

Advisor
Imam Kuswardayan, S.Kom., M.T.
Ridho Rahman Hariadi, S.Kom., M.Sc.

INFORMATICS DEPARTMENT
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2015

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

RANCANG BANGUN KECERDASAN BUATAN ADAPTIF PADA PERMAINAN TURN-BASED STRATEGY ADVANCE WARS

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Interaksi Grafis dan Seni
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
MUHAMMAD ARIF ROHMAN HAKIM
NRP. 5111100099

Disetujui oleh Pembimbing Tugas Akhir:

1. Imam Kuswardayan, S.Kom., M.T. .....
   NIP: 197612152003121001 (pembimbing 1)

2. Ridho Rahman Hariadi, S.Kom., M.Sc .....
   NIP: 198702132014041001 (pembimbing 2)

SURABAYA
JUNI, 2015

[Halaman ini sengaja dikosongkan]

RANCANG BANGUN KECERDASAN BUATAN ADAPTIF PADA PERMAINAN TURN-BASED STRATEGY ADVANCE WARS

Nama Mahasiswa : Muhammad Arif Rohman Hakim
NRP : 5111100099
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing I : Imam Kuswardayan, S.Kom., M.T.
Dosen Pembimbing II : Ridho Rahman Hariadi, S.Kom., M.Sc.

ABSTRAK

Penggunaan kecerdasan buatan telah dilakukan di berbagai bidang, salah satunya pada dunia permainan video komersial. Namun, kecerdasan buatan yang kebanyakan digunakan masih berupa kecerdasan buatan yang statis, antara terlalu sulit atau terlalu mudah. Hal ini dapat berakibat menurunnya minat pemain untuk memainkan permainan tersebut, sehingga berimbas pada menurunnya daya jual permainan tersebut di pasar.

Pada tugas akhir ini, penulis akan merancang dan membangun sebuah permainan strategi berbasis giliran (turn-based strategy) di mana sebuah kecerdasan buatan adaptif dengan implementasi fungsi evaluasi, yaitu fungsi yang dapat menghitung dan mengevaluasi setiap gerakan pemain terhadap keadaan permainan saat itu, digunakan untuk memainkan permainan ini. Kecerdasan buatan ini juga dapat diubah tingkat kesulitannya.

Dari hasil pengujian, kecerdasan buatan yang dibangun mampu bermain melawan pemain manusia dan kecerdasan buatan lain sesuai dengan tingkat kesulitannya.

Kata kunci: Kecerdasan Buatan, Kecerdasan Buatan Adaptif, Permainan Strategi Berbasis Giliran, Permainan Video, Fungsi Evaluasi.

[Halaman ini sengaja dikosongkan]

DESIGN AND IMPLEMENTATION OF ADAPTIVE ARTIFICIAL INTELLIGENCE FOR TURN-BASED STRATEGY GAME ADVANCE WARS

Name : **Muhammad Arif Rohman Hakim**
NRP : **5111100099**
Major : **Informatics Department, FTIf-ITS**
Advisor I : **Imam Kuswardayan, S.Kom., M.T.**
Advisor II : **Ridho Rahman Hariadi, S.Kom., M.Sc.**

ABSTRACT

The application of artificial intelligence had been used in various field, one of those fied was commercial video games. However, the majority of artificial intelligences used in those video games are still static artificial intelligences, which either too hard or too easy. This can cause the decline of player's desire to replay the game, which can impact the game's sales in market.

In this final project, the author will design and develop a turn-based strategy game where an adaptive artificial intelligence with the implementation of evaluation function, a function that can calculate dan evaluate every player's movement against current game condition, used to play this game. This artificial intelligence's diffculty also can be changed.

From the test results, the developed artificial intelligence can play against human player and other artificial intelligence based on its difficulty setting.

Keywords: Artificial Intelligence, Adaptive Artificial Intelligence, Turn-Based Strategy, Video Games, Evaluation Function.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Dengan menyebut nama Allah SWT yang Maha Pengasih lagi Maha Panyayang, kami panjatkan puja dan puji syukur atas kehadirat-Nya, yang telah melimpahkan rahmat, hidayah, dan inayah-Nya kepada kami, sehingga penulis dapat menyelesaikan tugas akhir ini dengan baik.

Penulis ingin menyampaikan rasa hormat dan terima kasih yang setinggi-tingginya kepada pihak-pihak yang telah membantu penulis dalam penyelesaian tugas akhir ini, terutama kepada:

1. Bapak Daryoso Basuki dan Ibu Sri Muningsih, orang tua penulis, yang selalu memberikan dukungan dan semangat baik dalam bantuan finansial maupun dorongan positif agar penulis mampu untuk menyelesaikan tugas akhir dengan benar dan tepat waktu, serta memberikan doa yang terus dikirimkan agar penyelesaian tugas akhir berjalan dengan lancar.
2. Adikku Muhammad Naufal Misbachuddin, yang selalu memberikan semangat dan memberikan doa agar penulis mampu menyelesaikan Tugas Akhirnya.
3. Ibu Umi Laili Yuhana, S.Kom., M.Kom. sebagai dosen wali penulis yang turut memberikan saran dan menuntun penulis dalam menentukan mata kuliah yang akan diambil pada pembelajaran di Jurusan Teknik Informatika ITS ini.
4. Bapak Imam Kuswardayan, S.Kom., M.T. dan Bapak Ridho Rahman Hariadi, S.Kom., M.Sc. yang telah bersedia untuk menjadi dosen pembimbing tugas akhir sehingga penulis dapat mengerjakan tugas akhir dengan arahan dan bimbingan yang baik dan jelas.
5. Teman-teman Mahasiswa Teknis Informatika 2011 yang telah berjuang bersama-sama selama menempuh pendidikan di Jurusan ini.

6. Teman-teman seperjuangan dari Pekalongan, Muhammad Adhijaya Saputra dan Muhammad Harun Fahad yang telah menemani penulis dikala suka dan duka.
7. Teman-teman asrama mahasiswa ITS khususnya Mas Fadli, Mas Zohan, Mas Arif, Mas Zaini, dan Dede yang telah memberikan inspirasi tersendiri bagi penulis.
8. Aldo Kelvianto Wachyudi sebagai teman sekamar dan teman dekat penulis saat masih menjadi mahasiswa baru hingga sekarang.
9. Teman-teman di Lab IGS, Didik Purwanto, Rifi Febrio Anggriawan, Bestama Abhi Priambadha, Askary Muhammad, M Iqbal Rustamadji dan teman-teman lain sebagai teman seperjuangan pengerjaan tugas akhir RMK IGS.
10. Teman-teman Lab Microsoft Mobility Lab, khususnya Lutfhan dan Ujik sebagai teman seperjuangan pengerjaan tugas akhir, serta Mas Yoza, Mas Ruka, Varis, Bilfash, Fany dan teman-teman yang lain yang selalu memberikan dukungan dan hiburan tersendiri kepada penulis.
11. Aldy Syahdeini, Luthfan Aufar Ramadhan, dan Uswatun Hasana Kunio sebagai teman seperjuangan Mobile Games Developer War 5 dimana penulis untuk pertama kalinya menjadi juara dalam ajang perlombaan IT selama berada di Jurusan Teknik Informatika.
12. Sandy Akbar Dewangga sebagai teman seperjuangan dalam perlombaan IWIC 8.
13. Teman-teman Sunday Sharing with Loperman (SSL), Hayam, Askary, Rezza, Ade, Hashfi, Ujik, Mas Aldy, Mas Sukma, dan teman-teman yang lain yang telah mengajak penulis dalam hal kebaikan.

14. Mahardhika Maulana sebagai rekan Kerja Praktek sekaligus teman dekat penulis.
15. Serta pihak-pihak lain yang turut membantu penulis baik secara langsung maupun tidak, yang namanya tidak penulis sebutkan disini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini, mohon maaf apabila ada kesalahan dan kata-kata yang dapat menyinggung perasaan. Penulis berharap tugas akhir ini dapat menjadi inspirasi bagi perkembangan teknologi permainan video di Indonesia.

Surabaya, Juni 2015

Penulis

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

LEMBAR PENGESAHAN vii
ABSTRAK ix
ABSTRACT xi
KATA PENGANTAR xiii
DAFTAR ISI xvii
DAFTAR GAMBAR xxi
DAFTAR TABEL xxv
DAFTAR KODE SUMBER xxvii
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Permasalahan 2
1.3 Batasan Permasalahan 2
1.4 Tujuan 3
1.5 Manfaat 3
BAB II DASAR TEORI 5
2.1 Kecerdasan Buatan 5
2.1.1 Ruang Lingkup Kecerdasan Buatan Pada Aplikasi Permainan Komersial 6
2.1.2 Kecerdasan Buatan Adaptif 7
2.1.3 Fungsi Evaluasi 9
2.2 Permainan *Turn-Based Strategy* 10
2.2.1 Rush Strategy 12
2.2.2 Unit Offence Strategy 12
2.2.3 Defensive Strategy 12

2.2.4 Balance Strategy 12
2.3 *Advance Wars* 12
2.4 A* Pathfinding Algorithm 14
2.5 Cocos2d-x 16
BAB III ANALISIS DAN PERANCANGAN SISTEM 17
3.1 Analisis Sistem 17
3.1.1 Deskripsi Umum Aplikasi Permainan 17
3.1.2 Spesifikasi Kebutuhan Aplikasi Permainan 18
3.1.3 Identifikasi Pengguna 19
3.2 Perancangan Sistem 19
3.2.1 Perancangan Objek Permainan 19
3.2.2 Perancangan Aturan Main Permainan 29
3.2.3 Perancangan Peta Permainan 39
3.2.4 Perancangan Antarmuka Permainan 39
3.2.5 Perancangan Kecerdasan Buatan 47
BAB IV IMPLEMENTASI SISTEM 63
4.1 Lingkungan Pengembangan 63
4.1.1 Lingkungan Pembangunan Perangkat Keras 63
4.1.2 Lingkungan Pembangunan Perangkat Lunak 63
4.2 Implementasi Antarmuka 64
4.2.1 Implementasi Antarmuka Layar *Main Menu* 64
4.2.2 Implementasi Antarmuka Layar *Battle* 65
4.2.3 Implementasi Antarmuka Layar *Change Turn* 66
4.2.4 Implementasi Antarmuka Layar Pause 67
4.2.5 Implementasi Antarmuka Window Buy Unit 67

4.2.6 Implementasi Antarmuka Window Game Over ..68
4.3 Implementasi Objek Permainan 69
4.4 Implementasi Aturan Main Permainan 70
4.4.1 Implementasi Tahap Persiapan 70
4.4.2 Implementasi Fase Siaga (*Standby Phase*).....72
4.4.3 Implementasi Membeli Unit.....73
4.4.4 Implementasi Memilih Unit 75
4.4.5 Implementasi Menggerakkan Unit.....77
4.4.6 Implementasi Menyerang Unit Lawan.....79
4.4.7 Implementasi Mengambilalih Bangunan.....81
4.5 Implementasi Kecerdasan Buatan Adaptif.....83
4.5.1 Implementasi Pembelian Unit 83
4.5.2 Implementasi Pergerakan dan Aksi Unit.....84
4.5.3 Implementasi Fungsi Evaluasi.....85
4.5.4 Implementasi Pengaturan Tingkat Kesulitan.....87
4.6 Implementasi Kecerdasan Buatan Lawan 88
4.6.1 Implementasi Pembelian Unit 88
4.6.2 Implementasi Pergerakan dan Aksi Unit *AI_Rush* 89
4.6.3 Implementasi Pergerakan dan Aksi Unit *AI_UnitOffence* 90
4.6.4 Implementasi Pergerakan dan Aksi Unit *AI_Defensive*.....91
4.6.5 Implementasi Pergerakan dan Aksi Unit *AI_Balance* 92
BAB V PENGUJIAN DAN EVALUASI 95
5.1 Lingkungan Pembangunan.....95

5.2 Skenario Pengujian 95
5.2.1 Pengujian Melawan Kecerdasan Buatan Lain 96
5.2.2 Pengujian Melawan Pemain Manusia 105
BAB VI KESIMPULAN DAN SARAN 109
6.1 Kesimpulan 109
6.2 Saran 110
DAFTAR PUSTAKA 111
LAMPIRAN A KODE SUMBER 117

DAFTAR TABEL

Tabel 3.1 Attribut Utama Objek Terrain.....20
Tabel 3.2 Terrain Pada Permainan.....20
Tabel 3.3 Attribut Utama Objek Unit 22
Tabel 3.4 Unit Pada Permainan 24
Tabel 3.5 Primary Damage.....25
Tabel 3.6 Secondary Damage.....25
Tabel 3.7 Attribut Utama Objek *Building*.....26
Tabel 3.8 Attribut Utama Objek Player 28
Tabel 5.1 Statistik Hasil Pengujian AI_Adaptive *Fair* Melawan AI_Rush 98
Tabel 5.2 Statistik Hasil Pengujian AI_Adaptive Melawan AI_UnitOffence.....101
Tabel 5.3 Statistik Hasil Pengujian AI_Adaptive *Hard* Melawan AI_Defensive.....101
Tabel 5.5 Statistik Hasil Pengujian AI_Adaptive Melawan AI_Balance.....104
Tabel 5.6 Tabel Daftar Penguji.....105
Tabel 5.7 Statistik Hasil Pengujian *AI_Adaptive* Melawan Pemain Manusia.....108

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

Gambar 2.1 Tampilan dari Permainan *Advance Wars* 13
Gambar 2.2 Ilustrasi Algoritma A* 15
Gambar 3.1 Finite State Machine Mekanisme Fase Siaga 30
Gambar 3.2 Finite State Machine Membeli Unit.....31
Gambar 3.3 Ilustrasi Pemilihan Unit 32
Gambar 3.4 Ilustrasi Pemilihan Unit Dengan Penghalang 32
Gambar 3.5 Finite State Machine Menggerakkan Unit.....33
Gambar 3.6 Finite State Machine Menyerang Unit Lawan.....34
Gambar 3.7 Finite State Machine Mengambilalih Bangunan 36
Gambar 3.8 Finite State Machine Permainan.....38
Gambar 3.9 Desain Peta Permainan.....39
Gambar 3.10 Tampilan Antarmuka Layar *Main Menu*.....40
Gambar 3.11 Tampilan Antarmuka Layar *Battle* 41
Gambar 3.12 Tampilan Antarmuka Layar *Change Turn* 43
Gambar 3.13 Tampilan Antarmuka Layar *Pause* 44
Gambar 3.14 Tampilan Antarmuka *Window Buy Unit* 45
Gambar 3.15 Tampilan Antarmuka *Window Game Over* 46
Gambar 3.16 FSM Mekanisme Pembelian Unit AI_Adaptive....48
Gambar 3.17 FSM Mekanisme Pergerakan Unit AI_Adaptive...50
Gambar 3.18 FSM AI_Adaptive.....55
Gambar 3.19 FSM Mekanisme Pembelian Unit Kecerdasan Buatan Lawan.....57
Gambar 3.20 FSM Mekanisme Pergerakan Unit AI_Rush 57
Gambar 3.21 FSM Mekanisme Pergerakan Unit AI_UnitOffence 59
Gambar 3.22 FSM Mekanisme Pergerakan Unit AI_Defensive .60
Gambar 3.23 FSM Mekanisme Pergerakan Unit AI_Balance.....61
Gambar 4.1 Tampilan Implementasi Antarmuka Layar *Main Menu* 64
Gambar 4.2 Tampilan Implementasi Antarmuka *Main Menu* Saat Memilih Tingkat Kesulitan 65
Gambar 4.3 Tampilan Implementasi Antarmuka Layar *Battle*....66
Gambar 4.4 Tampilan Implementasi Layar *Change Turn*.....66

Gambar 4.5 Tampilan Implementasi Antarmuka Layar *Pause* ...67
Gambar 4.6 Tampilan Implementasi *Window Buy Unit* 68
Gambar 4.7 Tampilan Implementasi Layar *Game Over*.....68
Gambar 4.8 Tampilan Implementasi Objek *Terrain*.....69
Gambar 4.9 Tampilan Implementasi Objek Unit.....70
Gambar 4.10 Tampilan Implementasi Objek *Building* 70
Gambar 4.11 Tampilan Implementasi Fase Siaga 73
Gambar 4.12 Tampilan Implementasi Membeli Unit 75
Gambar 4.13 Tampilan Implementasi Memilih Unit.....77
Gambar 4.14 Tampilan Implementasi Menggerakkan Unit 79
Gambar 4.15 Tampilan Implementasi Menyerang Unit Lawan ..79
Gambar 4.16 Tampilan Implementasi Mengambilalih Bangunan 81
Gambar 5.1 Histogram Hasil Pengujian AI_Adaptive *Fair* Melawan AI_Rush 97
Gambar 5.2 Histogram Hasil Pengujian AI_Adaptive *Hard* Melawan AI_Rush 97
Gambar 5.3 Histogram Hasil Pengujian AI_Adaptive *Easy* Melawan AI_Rush 98
Gambar 5.4 Histogram Hasil Pengujian AI_Adaptive *Fair* Melawan AI_UnitOffence.....99
Gambar 5.5 Histogram Hasil Pengujian AI_Adaptive *Hard* melawan AI_UnitOffence 100
Gambar 5.6 Histogram Hasil Pengujian AI_Adaptive *Easy* Melawan AI_UnitOffence.....100
Gambar 5.7 Histogram Hasil Pengujian AI_Adaptive *Hard* Melawan AI_Defensive 102
Gambar 5.8 Histogram Hasil Pengujian AI_Adaptive *Fair* Melawan AI_Balance.....103
Gambar 5.9 Histogram Hasil Pengujian AI_Adaptive *Hard* Melawan AI_Balance.....103
Gambar 5.10 Histogram Hasil Pengujian AI_Adaptive *Easy* Melawan AI_Balance.....104
Gambar 5.11 Histogram Hasil Pengujian AI_Adaptive *Fair* Melawan Pemain Manusia 106

Gambar 5.12 Histogram Hasil Pengujian AI_Adaptive Hard Melawan Pemain Manusia 107
Gambar 5.13 Histogram Hasil Pengujian AI_Adaptive Easy Melawan Pemain Manusia 107

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

Kode Sumber 4.1 Implementasi Tahap Persiapan 71
Kode Sumber 4.2 Implementasi Fase Siaga 73
Kode Sumber 4.3 Implementasi Membeli Unit 75
Kode Sumber 4.4 Implementasi Memilih Unit 77
Kode Sumber 4.5 Implementasi Menggerakkan Unit 79
Kode Sumber 4.6 Implementasi Menyerang Unit Lawan 81
Kode Sumber 4.7 Implementasi Mengambilalih Bangunan 82
Kode Sumber 4.8 Implementasi Pembelian Unit *AI_Adaptive* 84
Kode Sumber 4.9 Implementasi Pergerakan dan Aksi Unit *AI_Adaptive* 85
Kode Sumber 4.10 Implementasi Fungsi Evaluasi *AI_Adaptive* 87
Kode Sumber 4.11 Implementasi Pengaturan Tingkat Kesulitan *AI_Adaptive* 88
Kode Sumber 4.12 Implementasi Pembelian Unit Kecerdasan Buatan Lawan 89
Kode Sumber 4.13 Implementasi Pergerakan dan Aksi Unit *AI_Rush* 90
Kode Sumber 4.14 Implementasi Pergerakan dan Aksi Unit *AI_UnitOffence* 91
Kode Sumber 4.15 Implementasi Pergerakan dan Aksi *AI_Defensive* 92
Kode Sumber 4.16 Implementasi Pergerakan dan Aksi Unit *AI_Balance* 93
Kode Sumber 7.1 Implementasi Objek *Terrain* 117
Kode Sumber 7.2 Implementasi Objek Unit 120
Kode Sumber 7.3 Implementasi Objek *Building* 121
Kode Sumber 7.4 Implementasi Memilih Unit 123
Kode Sumber 7.5 Implementasi Menggerakkan Unit 126
Kode Sumber 7.6 Implementasi Pergerakan dan Aksi Unit *AI_Adaptive* 127
Kode Sumber 7.7 Implementasi Pengaturan Tingkat Kesulitan 129
Kode Sumber 7.8 Implementasi Pergerakan dan Aksi Unit *AI_Rush* 131

Kode Sumber 7.9 Implementasi Pergerakan dan Aksi Unit AI_UnitOffence.....133

[Halaman ini sengaja dikosongkan]

xxx

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir. Penjelasan bab ini akan dibagi menjadi beberapa subbab yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan.

1.1 Latar Belakang

Artificial Intelligence (AI) atau kecerdasan buatan adalah salah satu teknologi yang banyak diterapkan dalam berbagai bidang di dunia nyata mulai dari diagnosa medis, bursa saham, pengendali robot, penginderaan jauh, dan permainan video (*video game*). Kecerdasan buatan dalam permainan video paling banyak digunakan dalam mengendalikan *Non-Playable Character* (NPC), salah satunya untuk mengendalikan unit dalam permainan *turn-based strategy* (TBS). Banyak peneliti memperkenalkan kecerdasan buatan yang menjamin dapat mengalahkan lawannya. Dalam hal ini, ditujukan untuk permainan papan (*Board Game*) seperti Catur. Akan tetapi, kecerdasan buatan yang selalu mengalahkan pemain manusia sangat membuat frustasi bagi pemain yang melawannya. Bagi permainan video TBS komersial, hal ini sangat mudah mengakibatkan pemain untuk meninggalkan sebuah permainan dan tidak memainkannya lagi. Hal ini berdampak pada rusaknya reputasi permainan video tersebut, dan akhirnya penurunan penjualan permainan tersebut. Sebuah kecerdasan buatan yang diperuntukkan pada permainan video strategi komersial seharusnya sebuah kecerdasan buatan yang menyediakan tantangan yang masuk akal terhadap pemain manusia. Dengan kata lain, sebuah kecerdasan buatan yang kemampuannya sebanding dengan pemain manusia yang dilawannya. Oleh karena itu, tugas akhir ini akan membahas tentang penggunaan kecerdasan buatan adaptif yang dapat melawan pemain manusia secara seimbang dalam sebuah

permainan strategi berbasis giliran. Untuk mencapai hal ini, penulis menggunakan fungsi evaluasi untuk mengatur kecerdasan buatan tersebut ketika bermain melawan pemain manusia.

Dalam tugas akhir ini, penulis akan membuat sebuah aplikasi permainan *turn-based strategy* sendiri menggunakan informasi dan aturan berdasarkan serial permainan *Advance Wars* buatan Nintendo. Tugas akhir ini akan mengimplementasikan dan menguji kecerdasan buatan ini dengan memainkannya melawan pemain manusia dan beberapa script kecerdasan buatan lain.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana rancangan aturan main dalam permainan *turn-based strategy?*
2. Bagaimana cara membuat kecerdasan buatan yang dapat melawan pemain manusia secara seimbang dalam *turn-based strategy?*
3. Bagaimana cara mengatur tingkat kesulitan pada kecerdasan buatan ini?
4. Bagaimana performa kecerdasan buatan ini melawan kecerdasan buatan yang umum digunakan di permainan *turn-based strategy* yang lain?

1.3 Batasan Permasalahan

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. Aplikasi permainan hanya akan terdiri dari satu pertempuran (*battle*) yang dapat dimainkan berulang-ulang.
2. Pemain hanya dapat memilih peta permainan yang telah disediakan.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah:

1. Membuat aplikasi permainan dengan aturan *permainan turn-based strategy*.
2. Mengimplementasikan kecerdasan butan adaptif pada permainan *turn-based strategy*.

1.5 Manfaat

Manfaat dari hasil pembuatan tugas akhir ini antara lain :

1. Mengimplementasikan kecerdasan buatan adaptif dalam permainan *turn-based strategy* sehingga dapat dihasilkan sebuah permainan video *turn-based strategy* yang sesuai dengan kebutuhan permainan video komersial serta dapat dinikmati oleh pemain.
2. Dapat dijadikan referensi bagi penelitian tugas akhir (TA) yang lain.
3. Dapat dikembangkan lebih jauh menjadi permainan video komersial.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai dasar teori yang menjadi dasar pembuatan tugas akhir ini. Pokok permasalahan yang akan di bahas di bab ini adalah kajian mengenai permainan strategi berbasis giliran, Algoritma pencarian A*, Cocos2d-x, dan metode yang digunakan untuk membangun kecerdasan buatan.

2.1 Kecerdasan Buatan

Dalam sejarah, manusia telah membuat komputer untuk menyelesaikan bermacam-macam permasalahan seperti: aritmatika, pengurutan (*sorting*), pencarian (*searching*) dan sebagainya. Namun, ada beberapa hal di mana komputer tidak dapat melakukannya dengan baik, yang kadang manusia anggap sepele, seperti mengenali wajah, berbicar dengan bahasa manusia, memutuskan apa yang akan dilakukan, dan berkreasi. Untuk mengatasi hal-hal tersebut lahirlah cabang ilmu kecerdasan buatan (*artificial intelligence*). Livington menyatakan bahwa kecerdasan buatan adalah tentang bagaimana komputer dapat melakukan tugas berpikir yang mampu dilakukan oleh manusia dan hewan[1].

Kecerdasan buatan adalah salah satu cabang baru dalam ilmu sains dan rekayasa. Saat ini kecerdasan buatan telah digunakan dalam banyak bidang dalam kehidupan manusia, mulai dari fungsi umum seperti persepsi dan pengambilan keputusan, sampai masalah yang spesifik seperti pembuktian rumus matematika dan diagnosa penyakit.

Russel mengategorikan definisi kecerdasan buatan menjadi empat dimensi[2] yaitu:

1. Berpikir manusiawi (*thinking humanly*)
   Usaha untuk membuat komputer berpikir, mesin dengan pikiran, secara penuh dan harfiah. Aktivitas yang berhubungan dengan proses berpikir manusia, aktivitas

seperti pengambilan keputusan, penyelesaian masalah, dan belajar.

2. Berpikir rasional (*thinking rationally*)
   Pembelajaran tentang kemampuan mental melalui penggunaan model komputasional. Pembelajaran tentang komputasi yang memungkinkannya untuk melihat, mempertimbangkan, dan bertindak.
3. Bertindak manusiawi (*acting humanly*)
   Seni membuat mesin yang melakukan fungsi yang membutuhkan kecerdasan saat digunakan oleh orang. Pembelajaran tentang bagaimana membuat komputer melakukan sesuatu di mana, pada saat itu, orang akan melakukannya lebih baik.
4. Bertindak rasional (*acting rationally*)
   Pembelajaran tentang agen cerdas. Berhubungan dengan tingkah laku cerdas dalam sebuah penelitian.

Menurut Russel, tujuan riset mengenai kecerdasan buatan adalah penalaran (*reasoning*), pengetahuan (*knowledge*), perencanaan (*planning*), pembelajaran (*learning*), pemrosesan bahasa natural (*natural language processing*), persepsi (*perception*) dan kemampuan untuk memanipulasi objek.

2.1.1 Ruang Lingkup Kecerdasan Buatan Pada Aplikasi Permainan Komersial

Kecerdasan buatan telah digunakan dalam aplikasi permainan video (video games) sejak awal perkembangan permainan itu sendiri. Permainan *Pong*, buatan Al Alcorn untuk perusahaan Atari tahun 1972, yang merupakan permaian video yang mengawali bisnis di industri permainan video, telah menggunakan kecerdasan buatan sederhana untuk menggerakkan balok pemain ke dua arah berlawanan[3].

Kecerdasan buatan dalam permainan video telah dapat melakukan tindakan yang lebih pintar sejak munculnya permainan

video Pacman, buatan Midway Games West tahun 1979. Pacman mengimplementasikan teknik kecerdasan buatan sederhana yang disebut *state machine*. Setiap monster dalam permainan tersebut memiliki salah satu dari dua keadaan atau *state*, yaitu antara mengejar pemain atau menghindari pemain jika pemain mengambil *power-up*.

Pada pertengahan 90-an, *game* berjenis strategi beredar luas di pasaran. Pada jenis permainan ini, mulai digunakan teknik pencarian jejak (*pathfinding*) untuk mencari rute ke posisi tertentu.

Awal tahun 2000, permainan video telah menggunakan kecerdasan buatan yang kompleks, dengan penggunaan *neural network* untuk melakukan pengambilan keputusan.

Meski kecerdasan buatan dalam industri permainan video telah berkembang pesat, sebagian besar permainan video di pasaran masih menggunakan kecerdasan buatan sederhana seperti yang digunakan *Pacman* dikarenakan kecerdasan buatan yang seperti itu sudah mencukupi kebutuhan.

Kecerdasan buatan pada permainan video modern, menurut Livington, digunakan untuk menangani tiga kebutuhan dasar dalam permainan, yaitu kemampuan untuk menggerakkan karakter, kemampuan untuk mengambil keputusan tentang ke mana akan bergerak, dan kemampuan untuk berpikir secara taktis atau strategis.

2.1.2 Kecerdasan Buatan Adaptif

Kecerdasan buatan berkualitas tinggi telah menjadi nilai jual yang penting pada permainan video[4]. Walaupun begitu, pemain masih lebih memilih untuk bermain melawan sesama pemain manusia dibandingkan dengan komputer. Hal ini dikarenakan komunitas pemain permainan video merasakan bahwa kualitas dari kecerdasan buatan pada permainan video masih dianggap rendah[5]. Sebagai contoh, kemampuan kecerdasan buatan pada permainan kompleks seperti *Go* (sejenis permainan catur dari Jepang) jauh melebihi kemampuan manusia[6]. Namun, ada

beberapa situasi di mana kecerdasan buatan yang menghibur dengan nilai bermain yang tinggi masih dibutuhkan. Contohnya situasi di mana tidak tersedianya koneksi jaringan internet (misal di tempat umum atau dalam penerbangan), atau cukup karena pemain sedang ingin bermain sendiri.

Sebuah permainan komersial akan dimainkan oleh banyak pengguna dengan gaya dan strategi bermain yang berbeda-beda. Maka dari itu, sebuah kecerdasan buatan yang statis akan kesulitan untuk dapat melayani semua gaya bermain dari pemain manusia. Di sisi lain, sebuah kecerdasan buatan adaptif untuk aplikasi permainan memiliki potensi untuk membuat pengalaman bermain yang berbeda untuk pemain yang berbeda pula, dengan demikian menambah nilai dan kemampuan untuk dimainkan kembali (*replayability*) pada sebuah aplikasi permainan. Sebuah penelitian oelh Hagelback dan Johansson menunjukkan bahwa pemain lebih menikmati bermain dengan lawan yang beradaptasi dengan kemampuan bermainnya[7].

Kecerdasan buatan adaptif pada aplikasi permainan merujuk pada sebuah pemain yang dikendalikan oleh komputer secara dinamis yang menyesuaikan kemampuan bermainnya sebagai respon terhadap lawannya[8]. Riset mengenai metode dan mekanisme kecerdasan buatan adaptif dalam permainan video telah luas dilakukan oleh para peneliti. Beberapa peneliti menggunakan metode kecerdasan komputasional (*computational intelligence*) dengan *neural network*. Bergsma dan Spronck mengimplementasikan metode yang dinamakan *Allocation and Decomposition Architecture for Perfroming Tactical AI* (ADAPTA) yang dapan mempelajari dan mengalahkan musuh pada permainan strategi berbasis giliran[9]. Bryant dan Miikkulainen menggunakan metode kecerdasan komputasional menggunakan neuroevolution pada permainan strategi yang dapat menyesuaikan strategi pemainnya[10]. Stanley menggunakan metode bernama *real-time version of NeuroEvolution of Augmenting Topologies* (rtNEAT) yang memungkinkan agen pada

sebuah permainan untuk mengadopsi dan memperbaiki diri pada saat permainan berlangsung[11].

Dalam sejarah penggunaannya, metode kecerdasan komputasional menggunakan *neural network* untuk membuat kecerdasan buatan adaptif pada aplikasi permainan video memiliki beberapa kekurangan, diantaranya yaitu sangat susah untuk mengidentifikasi input dan output pada *neural network* yang masuk akal dengan konteks permainan. Selain itu, kekurangan lain adalah sebagian besar *neural network* belajar melalui teknik yang disebut "*supervised learning*" atau "belajar yang diawasi", yang berarti komputer harus diajari terlebih dahulu menggunakan set yang disebut *"training"* tentang bagaimana komputer harus mengeluarkan output yang sesuai jika dimasukkan input tertentu, di mana hal tersebut memerlukan umpan balik yang konstan dan terus-menerus dari pemain. Walaupun membangun *neural network* yang dapat belajar tanpa diawasi (*unsupervised learning*) mungkin untuk dilakukan, tidak ada jaminan bahwa komputer tidak akan "bertindak bodoh dan menjadi pemain yang benar-benar berdaya. Beberapa pengembang mencoba menghindari permasalahan ini dengan melatih *neural network* mereka secara eksklusif pada saat pengembangan sebelum aplikasi permainan tersebut dipasarkan. Hal ini memungkinkan kecerdasan buatan mereka berlatih melawan tim pengembang. Kelemahan dari cara ini adalah, tentu saja kecerdasan buatan tersebut tidak belajar apapun dari pemain nantinya[12].

2.1.3 Fungsi Evaluasi

Beberapa peneliti lain menemukan cara untuk menghindari kelemahan dari penggunaan *neural network,* yaitu dengan menggunakan sebuah fungsi evaluasi (*evaluation function*). Fungsi evaluasi adalah sebuah rumus atau formula yang digunakan untuk mengevaluasi sebuah tindakan sebelum tindakan itu dilakukan dengan menggunakan informasi yang berisi parameter-parameter tertentu. Bakkes[13], dalam publikasinya, menggunakan jumlah

unit lawan yang diukur menggunakan jangkauan penglihatan (*visibility range*). Strattman[14] dalam publikasinya, menggunakan fungsi evaluasi untuk mencari titik temu (*waypoint*) terbaik di dalam permaian *First-Person Shooter* (FPS). Sedangkan Potisartra dan Kotrajaras menggunakan fungsi evaluasi dalam permaiann *turn-based strategy*[15].

2.2 Permainan *Turn-Based Strategy*

Permainan *Turn-based strategy* atau strategi berbasis giliran di aplikasi permainan adalah permainan yang terinspirasi dengan permainan fisik papan. Permainan papan biasanya perlu menggunakan strategi yang baik agar dapat dimainkan. Selama bermain permainan startegi, setiap keputusan dan setiap gerakan yang dibuat adalah peluang untuk mendapat hasil yang berarti[16]. Keputusan diartikan sebagai situasi dimana pemain memiliki informasi yang cukup dan memadai pada satu waktu. Itu artinya dia tidak memiliki solusi alternatif selain apa yang sudah dimiliki saat ini[17].

Kelebihan permainan dengan sistem ini adalah tersedianya kontrol penuh terhadap bidak mereka dan tidak bergantung reaksi refleks saat bermain sehingga memungkinkan banyak strategi dan pilihan saat bermain. Karena itu perlu perencanaan dan atisipasi yang baik dari pemain. Selain itu pelacakan strategi musuh juga jauh lebih mudah dalam sistem berbasis giliran karena sifatnya saling bergantian.

Kekurangan di sistem terdapat pada desain yang buruk yang mencolok ketika pertempuran. Selain itu juga terdapat dalam animasi yang lambat. Namun kelemahan terbesar sistem ini adalah tidak adanya pertempuran yang menantang yang dapat membuang-buang waktu pemain. Beberapa kelemahan lainnya adalah sistem permainan ini tidak lebih trend dibandingkan yang lainnya.

Garetsh Davis memberikan saran agar permainan strategi berbasis giliran dapat dibuat dengan baik. Berikut ini adalah saran-saran pengembangan permainan berbasis giliran[18].

Pertama adalah efisiensi. Efisiensi diartikan sebagai banyaknya keputusan yang menarik saat permainan. Semakin banyak keputusan yang tersedia setiap detiknya maka semakin menarik. Jadi hal-hal sepele di luar keputusan penting harus dikurangi. Seperti animasi yang tidak penting, pengaturan kamera dan lain halnya yang tidak berhubungan dengan keputusan saat permainan.

Kedua adalah variasi dalam permainan. Permainan strategi yang bagus mampu menyediakan beragam situasi. Game multiplayer biasanya menawarkan potensi variasi yang lebih tinggi karena aksi musuh tidak dapat benar-benar diprediksi sehingga menciptakan pengalaman bermain bagi pemain. Selain itu adanya kondisi-kondisi random dapat mendukung munculnya variasi dalam permainan.

Ketiga adalah terpadu. Keterpaduan antar komponen dalam permainan menggambarkan keterkaitan mekanisme keputusan. Jika satu elemen terhubung ke banyak hal tentu akan mengarahkan ke banyak potensi keputusan di dalam sistem. Sementara hal-hal yang tidak berhubungan dengan mekanisme utama permainan dapat dihapuskan.

Keempat adalah keseimbangan. Keseimbangan tidak hanya fokus kepada keseimbangan elemen-elemen sistemik namun juga keseimbangan pada keputusan itu sendiri. Artinya keputusan-keputusan alternatif juga sama beratnya dengan keputusan lainnya. Sebagai contoh di dalam game perkelahian dimana terdapat dua jenis serangan yaitu serangan kuat dan serangan lemah. Jika ada dua faktor yang berpengaruh yaitu kecepatan dan tenaga, tentu jenis serangan kuat tidak boleh memiliki kelebihan dalam faktor kecepatan cukup faktor tenaga saja. Jika serangan kuat memiliki kelebihan dalam kecepatan tentu saja tidak perlu diberikan alternatif keputusan berupa serangan lemah.

Menurut Mark Newheiser, keseimbangan dalam bermain permainan strategi adalah jika tidak ada satu atau lebih strategi yang dapat mendominasi permainan[19]. Permainan yang memiliki satu atau lebih strategi yang mendominasi permainan

yang mengurangi tingkat kesenangan dari para pemainnya. Karena itu, permainan strategi harus dapat dimainkan dengan berbagai variasi cara.

Dalam bermain permainan turn-based strategy, ada beberapa strategi yag dapat digunakan. Beberapa strategi yang dipakai dalam bermain permaian *turn-based strategy* yang dipakai oleh Potisartra dan Kotrajaras dalam riset mereka antara lain:

2.2.1 Rush Strategy

Strategi ini dimulai dengan setiap unit mencari lawan yang jaraknya paling dekat dengan dirinya masing-masing. Kemudian setiap unit tersebut menyerang unit lawan sampai unit lawan tersebut kalah. Jika ada unit lawan lain yang lebih dekat, ganti target serangan ke unit lawan tersebut.

2.2.2 Unit Offence Strategy

Strategi ini dimulai dengan setiap unit mencari unit lawan secara random, kemudian terus menyerang unit lawan tersebut sampai unit lawan tersebut kalah, sebelum berganti target ke unit lawan lainnya.

2.2.3 Defensive Strategy

Strategi ini dimulai dengan setiap unit mencari unit kawan terdekat. Kemudian hanya menyerang jika di provokasi (diserang lebih dulu) oleh unit lawan.

2.2.4 Balance Strategy

Strategi ini merupakan gabungan dari *Unit Offence Strategy* dan *Defense Strategy*. Beberapa unit melakukan *Unit Offence Strategy*, dan unit-unit lainnya melakukan *Defensive Strategy*.

2.3 *Advance Wars*

Advance Wars adalah sebuah permainan video berjenis Turn-Based Strategy games yang dibuat oleh Nintendo. Dalam permaian *Advance Wars*, terdapat dua buah pihak yang mengendalikan sekumpulan pasukan atau unit di sebuah peta (*map*) berbentuk *array* 2-dimensi yang disebut *tilema*p. Sebuah *tilemap* terdiri dari kumpulan persegi yang disebut *tile*. Kedua pihak dapat menggerakkan aksi terhadap unit-unitnya dari satu *tile* ke *tile* lain secara bergantian. Setiap unit memiliki atribut *movement range*, *attack range, hit points,* dan *attack damage* yang berbeda-beda sesuai jenis unit-nya.

Gambar 2.1 Tampilan dari Permainan *Advance Wars*

Movement range adalah atribut yang menentukan seberapa jauh sebuah unit dapat bergerak, sedangkan *attack range* menetukan seberapa jauh sebuah unit dapat menyerang unit lain. Sebuah unit hanya dapat bergerak sejauh *movement range* miliknya. *Hitpoints* adalah atribut yang menentukan nyawa dari sebuah unit. Jika *hitpoints* bernilai nol, maka unit tersebut dinyatakan kalah dan dihilangkan dari peta permainan. *Attack damage* adalah atribut yang menentukan jumlah pengurangan *hit points* lawan ketika sebuah unit menyerang unit lain. Ketika sebuah unit masuk dalam *attack range* sebuah unit, maka unit dapat

menyerang unit yang masuk dalam jangkauannya tersebut. Unit yang terkena serangan akan berkurang attribut *hitpoints*-nya sesuai dengan attribut *attack damage* unit penyerangnya.

Permainan selesai apabila salah satu pihak dapat mengalahkan semua unit lawannya atau memenuhi tujuan (*objective*) tertentu, seperti mengalahkan unit tertentu, atau menangkap bangunan lawan.

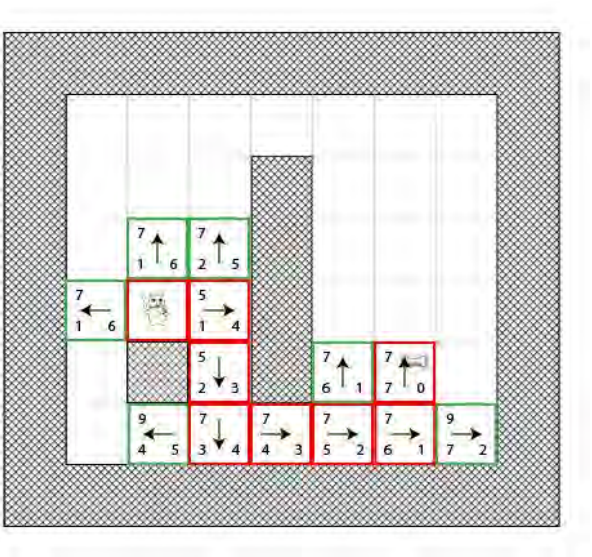
2.4 A* Pathfinding Algorithm

A* (*A-Star*) adalah sebuah algoritma untuk mencari jarak terpendek yang banyak digunakan di berbagai aplikasi. Dalam aplikasi permainan ini, algoritma A* digunakan untuk mencari rute terpendek ketika sebuah unit akan bergerak ke suatu titik sesuai dengan jangkauan gerakannya. Algoritma A* akan menghitung setiap *tile* yang akan dilalui oleh sebuah unit berdasarkan persamaan 2.1.

$$F(x,y) = G(x,y) + H(x,y) \quad (2.1)$$

Di mana F(x,y) adalah skor untuk setiap *tile.* G(x,y) adalah biaya (*cost)* yang diperlukan dari titik start ke titik saat ini. Biaya dalam hal ini adalah jumlah *tile.* Jadi untuk *tile* yang bersebelahan dengan titik start akan bernilai 1, namun akan bertambah jika *tile* tersebut lebih jauh dari titik start. H(x,y) adalah fungsi *heuristic* yang menghitung skor sebuah *tile* ke titik tujuan. H(x,y) disebut fungsi *heuristic* karena hanya menghitung perkiraan biaya sebuah *tile*.

Diluar fungsi-fungsi tersebut, algoritma A* membutuhkan dua buah *list,* yaitu *open list* dan *closed list.* ***Open list*** adalah list yang digunakan untuk menyimpan *tile* yang akan dihitung skornya. ***Closed list*** adalah list yang digunakan untuk menyimpan *tile* yang sudah dihitung skornya.



Gambar 2.2 Ilustrasi Algoritma A*

Algoritma A* akan mencari *tile* mana saja yang mengarah ke jarak terdekat ke suatu titik dengan menjalankan perintah berikut secara berulang :

1. Ambil *tile* dalam *open list* yang memiliki skor terkecil. Sebut *tile* ini *tile* S.
2. Buang *tile* S dari *open list* dan masukkan ke dalam *closed list*.
3. Lakukan hal berikut untuk setiap *tile* T di dalam tetangga S yang dapat dilalui:
   a) Jika T ada di dalam *closed list*, abaikan.
   b) Jika T tidak dalam berada *open list*, masukkan dan hitung skornya.
   c) Jika T berada dalam *open list,* hitung nilai F(x,y) dan bandingkan apakah nilainya lebih rendah saat nilai F(x,y) yang digunakan untuk sampai ke sini. Jika ya, perbarui skornya dan perbarui skor *parent*-nya juga.

2.5 Cocos2d-x

Cocos2d-x adalah sebuah *game framework open source* yang diperuntukkan untuk mengembangkan permainan video di banyak *platform,* mulai dari Sistem operasi *desktop* seperti Windows, Linux, dan MacOS, sampai perangkat bergerak seperti Android, iOS, Windows Phone, dan Tizen. Cocos2d-x menggunakan C++ sebagai bahasa pemrogramannya dan Visual Studio untuk lingkungan pengembangannya (jika menggunkan sistem operasi Windows). Cocos2d-x telah digunakan untuk membuat banyak permainan populer, diantaranya *BADLAND, Castle Clash, LINE Let's Get Rich, Brave Frontier, dan Zenonia*[3]. Cocos2d-x dapat digunakan untuk membuat permainan 2D maupun 3D, namun lebih banyak digunakan untuk 2D karena banyaknya fitur yang mempermudah pengembang, diantaranya *SceneManager* dan dukungan untuk *tilemap*.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Bab ini membahas tahap analisis permasalahan dan perancangan dari sistem yang akan dibangun. Analisis permasalahan membahas permasalahan yang yang diangkat dalam pengerjaan tugas akhir. Analisis kebutuhan mencantumkan kebutuhan-kebutuhan yang diperlukan perangkat lunak. Selanjutnya dibahas mengenai perancangan sistem yang dibuat.

3.1 Analisis Sistem

Pada subbab berikut akan dijelaskan analisa pembuatan aplikasi permainan strategi giliran berbasis *Windows desktop*. Analisa yang dilakukan meliputi deskripsi umum perangkat lunak, alur proses pembuatan, kebutuhan fungsional dan non-fungsional sistem, arsitektur sistem, dan identifikasi pengguna.

3.1.1 Deskripsi Umum Aplikasi Permainan

Aplikasi yang akan dibangun adalah sebuah permainan *turn-based strategy* berdasarkan informasi, mekanisme, dan aturan yang diambil dari serial permainan *Advance War* buatan Nintendo. Dalam permainan ini, dua pihak saling bertempur satu sama lain dengan mengendalikan pasukan (disebut unit) dalam sebuah peta permaian berbentuk array 2 dimensi (disebut *tilemap*). Unit dibeli dari bangunan yang disebut *factory*. Pemain mengendalikan unitnya tiap giliran (*turn*). Pemain mendapat uang dari jumlah bangunan yang dikuasainya. Terdapat bangunan kota (*city*) yang dapat menyembuhkan unit. Pemain dapat mengambil alih bangunan milik pemain lain maupun yang netral dengan menggerakkan unitnya ke bangunan, dan memilih aksi "*capture"*. Tidak semua unit dapat mengambilalih bangunan. Pemain dinyatakan menang apabila dapat mengalahkan semua unit lawannya atau dapat mengambil alih markas (*headquarter*)

lawannya. Pemain dapat mengambil alih markas lawannya dengan menggerakkan unitnya ke markas musuh.

3.1.2 Spesifikasi Kebutuhan Aplikasi Permainan

Kebutuhan sistem yang akan dibuat ini melibatkan dua hal, yakni kebutuhan fungsional maupun kebutuhan non-fungsional. Dimana masing-masing berhubungan dengan keberhasilan dalam pembuatan aplikasi tugas akhir ini.

3.1.2.1 Kebutuhan Fungsional Aplikasi Permainan

Berdasarkan deskripsi umum sistem di subbab sebelumnya, maka dapat disimpulkan bahwa kebutuhan fungsional dari aplikasi ini adalah sebagai berikut:

a) Aplikasi mampu menjalankan aturan main permainan strategi berbasis giliran berdasarkan informasi, mekanisme, dan aturan yang diambil dari serial permainan *Advance War*s.
b) Aplikasi mampu menampilkan grafis dari semua asset gambar permainan yang dimilikinya.
c) Kecerdasan buatan yang akan dibuat mampu mengimbangi kemampuan pemain lawannya.
d) Pemain dapat memilih tingkat kesulitan kecerdasan buatan yang akan dilawannya. Kecerdasan buatan yang akan dibuat harus dapat menyesuaikan tingkat kesulitan yang dipilih pemain.

3.1.2.2 Kebutuhan Non-fungsional Aplikasi Permainan

Terdapat beberapa kebutuhan non-fungsional yang apabila dipenuhi, dapat meningkatkan kualitas dari permainan ini. Berikut daftar kebutuhan non-fungsional:

a) Kenyamanan
   Aplikasi permainan yang dibangun memiliki resolusi layar minimal 960x640 piksel agar aplikasi memiliki tampilan yang sesuai.
b) Performa
   Memori yang penuh dapat membuat aplikasi terhenti atau hang secara tiba-tiba sehingga perlu dipastikan bahwa perangkat komputer yang digunakan untuk memainkan aplikasi permainan ini memiliki ruang penyimpanan kosong sebesar minimal 100MB dan RAM minimal 1GB.

3.1.3 Identifikasi Pengguna

Dalam aplikasi tugas akhir ini, pengguna yang akan terlibat hanya terdapat satu orang saja, yakni pemain yang akan bermain permainan strategi berbasis giliran melawan kecerdasan buatan.

3.2 Perancangan Sistem

Tahap perancangan dalam subbab ini dibagi menjadi empat bagian, yaitu perancangan alur permainan, perancangan objek permainan, perancangan antarmuka permainan, dan perancangan kecerdasan buatan.

3.2.1 Perancangan Objek Permainan

Objek dalam permainan ini dibagi menjadi empat kategori: *terrain, unit, building*, dan *player*.

3.2.1.1 Terrain

Terrain adalah jenis *tile* atau "bidak" dalam permainan. Secara garis besar, *tile* ada dua jenis. Jenis pertama adalah *passable*, artinya terrain tersebut dapat dilewati oleh sebuah unit. Jenis kedua adalah *impassable*, artinya terrain tersebut tidak dapat dilewati oleh

sebuah unit. Namun tidak hanya itu, setiap terrain mempunyai atrribut-attribut lain. Untuk lebih jelasnya, semua attribut utama yang dimiliki terrain dapat dilihat di Tabel 3.1.

Tabel 3.1 Attribut Utama Objek Terrain

Nama	Tipe Data	Keterangan
terrain ID	Integer	ID unik tiap terrain.
***tile*Type**	String	Nama dari terrain ini
cover	Integer	Menentukan berapa terrain tersebut dapat melindungi unit yang berada di atasnya. Berpengaruh pada perhitungan *damage*.
movement Cost	Integer	Menentukan jumlah *tile* yang harus dikurangi ketika unit akan bergerak melewati terrain ini. 999 berarti terrain ini tidak dapat dilewati unit secara umum.
hScore	Integer	Skor fungsi *heuristic* yang digunakan pada fungsi *pathfinding* A*
fScore	Integer	Skor akhir *tile* yang digunakan pada fungsi *pathfinding* A*

Jenis-jenis *terrain* yang digunakan dalam aplikasi permainan ditunjukkan pada Tabel 3.2.

Tabel 3.2 Terrain Pada Permainan

terrainId	*tile*Type	cover	movementCost
0	*Road*	0	1
1	*Plains*	1	1
2	*Forest*	2	1

3	*Mountain*	4	999
4	*River*	0	999
5	*Bridge*	0	1
6	*Sea*	0	999
7	*City*	3	1
8	*Factory*	3	1
9	*HQ*	4	1
10	*Airport*	3	1
11	*Port*	3	1
12	*Impassable*	0	999
13	*Shoal*	0	1
14	*Reef*	0	1

3.2.1.2 Unit

Unit adalah bagian vital dalam permainan. Pemain dapat membeli unit dari bangunan pabrik (*factory*) yang dimilikinya. Unit digunakan pemain untuk menyerang unit lain maupun untuk mengambilalih bangunan. Tidak semua unit dapat mengambil alih bangunan.

Unit menyerang unit lain dan mengurangi nilai *healthPoint* (HP) unit lawannya berdasarkan nilai "*damage*" yang dimilikinya terhadap unit lawannya, setelah melewati fungsi perhitungan lagi. Terdapat dua buah tipe *damage*, yaitu *primary* dan *secondary*. *Primary damage* adalah *damage* yang dihasilkan jika menggunakan attribut "*ammunition*", sedangkan *secondary damage* tidak memerlukan *ammunition*. Tidak semua unit memiliki *ammunition* sehingga tidak semua unit memiliki attribut *primary damage*. Sebaliknya, tidak semua unit memiliki *secondary damage*. Secara *default*, unit akan menggunakan *primary damage* jika ia memilikinya. Beberapa unit dapat bergerak melewati *terrain* yang memiliki attribut *movementCost* sebesar 999 (tidak dapat dilewati). Kemampuan itu dilihat dari attribut "*movementCostOverride*" miliknya. Sebagai contoh, unit "*infantry*" dapat berjalan melewati *terrain* "*mountain*" (yang memiliki

attribut *movementCost* 999) karena ia memiliki attribut *movementCostOverride* untuk "*mountain*" sebesar 2. Setiap unit memiliki tipe attribut yang sama dengan nilai yang berbeda-beda. Attribut utama dari objek unit ditunjukkan pada Tabel 3.3.

Tabel 3.3 Attribut Utama Objek Unit

Nama	Tipe Data	Keterangan
unitId	Integer	ID unik setiap unit.
name	String	Nama dari unit ini, misal "tank" atau "attack helicopter".
cost	Integer	harga unit ini
movementPoints	Integer	Poin yang menyatakan seberapa jauh (dinyatakan dalam *tile*) unit ini dapat bergerak.
movementCostOverride	Array of Integer	*Array* yang berisi nilai untuk meng-*override* movementCost pada *tile*Tertentu. Misal unit "*infantry*" dapat bergerak di terrain "*mountain*" yang bernilai 999, karena memiliki nilai *override* untuk "mountain" sebesar 2.
hitPoints	Integer	Poin yang menyatakan tingkat "kesehatan" dari unit ini disingkat "HP". Menunjukkan nilai HP saat ini.

maxHitPoints	Integer	Poin yang menyatakan tingkat "kesehatan" dari unit ini disingkat "HP". Menunjukkan nilai HP saat maksimal unit ini.
fuel	Integer	Bahan bakar unit saat ini.
maxFuel	Integer	Bahan bakar maksimal unit ini.
ammunition	Integer	Amunisi unit saat ini.
maxAmmunition	Integer	Amunisi maksimal unit ini.
minAttackRange	Integer	Jarak serangan minimal unit ini (dinyatakan dalam satuan *tile*).
maxAttackRange	Integer	Jarak serangan maksimal unit ini (dinyatakan dalam satuan *tile*).
canActAfterMoving	Boolean	Menyatakan apakah unit ini dapat melakukan aksi setelah bergerak.
canCounter	Boolean	Menyatakan apakah unit ini dapat melakukan counter attack.
can*Capture*	Boolean	Menyatakan apakah unit ini dapat mengambilalih bangunan.
player	Integer	Pemain yang memiliki unit ini.

primaryDamage	Array of Integer	*Array* berisi damage untuk setiap jenis unit, jika menggunakan *ammunition*.
secondaryDamage	Array of Integer	*Array* berisi damage untuk setiap jenis unit, jika tidak menggunakan *ammunition*.

Daftar unit yang digunakan dalam aplikasi permainan ini ditunjukkan pada Tabel 3.4.

Tabel 3.4 Unit Pada Permainan

unitID	name	cost (x 1000)	movementPoints	maxHitpoints	maxFuel	maxAmmunition	minAttackRange	maxAttackRange	canActAfterMoving	canCounter	can*Capture*
0	*Infantry*	1	3	10	99	0	1	1	T	T	T
1	*Mech*	3	2	10	70	3	1	1	T	T	T
2	*Recon*	4	8	10	80	0	1	1	T	T	F
3	*APC*	5	6	10	70	0	1	1	1	F	F
4	*Anti-Air*	8	6	10	60	9	1	1	T	T	F
5	*Tank*	7	6	10	70	9	1	1	T	T	F
6	*Medium Tank*	16	5	10	50	8	1	1	T	T	F
7	*Neo Tank*	22	6	10	99	9	1	1	T	T	F
8	*Artillery*	6	5	10	50	9	2	3	F	F	F
9	*Rocket*	15	5	10	50	6	3	5	F	F	F
10	*Missiles*	12	4	10	50	6	3	5	F	F	F

Tabel *primary damage* dan *secondary damage* untuk setiap unit ditunjukkan pada Tabel 3.5 dan Tabel 3.6.

Tabel 3.5 Primary Damage

unit	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	85	75	65	55	15	15	70	85	85
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	105	105	60	50	45	25	10	5	50	55	55
5	0	0	85	75	65	55	15	15	70	80	85
6	0	0	105	105	105	85	55	45	105	105	105
7	0	0	125	125	115	105	75	55	115	125	125
8	90	85	80	70	75	70	45	40	75	80	80
9	95	90	90	80	85	80	55	50	80	85	90
10	0	0	0	0	0	0	0	0	0	0	0

Tabel 3.6 Secondary Damage

unit	0	1	2	3	4	5	6	7	8	9	10
0	55	45	12	14	5	5	1	1	15	25	25
1	65	55	18	20	6	6	1	1	32	35	35
2	70	65	35	45	4	6	1	1	45	55	28
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	75	70	40	45	6	6	1	1	45	55	30
6	105	95	45	45	10	8	1	1	45	55	35
7	125	115	65	45	17	10	1	1	65	75	55
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0

3.2.1.3 Building

Building adalah terrain yang menghasilkan uang setiap turn dan dapat diambilalih oleh pemain. *Building* telah tersedia dalam peta ketika permainan dimulai. Setiap *building* memiliki fungsi yang berbeda-beda, seperti mereparasi unit dan memproduksi unit. Attribut utama yang dimiliki oleh *building* ditunjukkan pada Tabel 3.7.

Tabel 3.7 Attribut Utama Objek *Building*

Nama	Tipe Data	Keterangan
***building*ID**	Integer	ID unik setiap bangunan yang membedakan jenis bangunan dengan bangunan lain
name	String	Nama dari bangunan ini.
canBe*Captured*	Boolean	Menyatakan apakah bangunan ini dapat di*capture* oleh pemain
***capture*Point**	Integer	Menyatakan berapa kali harus di-*capture* agar bangunan dapat dimiliki oleh pemain
max*Capture*Point	Integer	*capture*Point maksimal yang dimiliki bangunan ini
player	Integer	Pemain yang memiliki bangunan ini.(pemain

		memiliki nilai 1 atau 2, 0 berarti netral).

Dalam aplikasi permainan ini hanya akan terdapat tiga jenis *building*, yaitu: *Headquarter*, *Factory*, dan *City*.

3.2.1.3.1 Headquarter (HQ)

Headquarter, atau disingkat *HQ* berperan sebagai bangunan utama pemain dalam permainan. Mengambilalih *HQ* seorang pemain akan menyebabkan pemain tersebut langsung dinyatakan kalah, tidak peduli berapa sisa unit yang ada di peta permainan. Disamping hal ini, *HQ* berperan sama seperti *building* lain: *HQ* dapat menyembuhkan *HP* unit, serta mengisi amunisi (*ammunition*) dan bahan bakar (*fuel*). *HQ* juga memberikan 1000 uang (*gold*) kepada pemain setiap giliran kepada pemain yang memiliki.

3.2.1.3.2 Factory

Factory dapat memproduksi sebuah unit dengan biaya tertentu yang harus dibayar pemain yang memilikinya. Seperti *building* lain, *factory* juga dapat menyembuuhkan *HP* unit kawan, serta mengisi *ammunition* dan *fuel*. *Factory* juga menyumbang 1000 *gold* kepada pemain yang memiliki setiap gilirannya. *Factory* yang diambilalih lawan dapat berakibat fatal, karena lawan dapat langsung memproduksinya di garis pertahanan lawan, yang biasanya dekat dengan *HQ*.

3.2.1.3.3 City

City merupakan *building* yang hanya memiliki fungsi dasar: menyembuhkan 2 HP unti kawan, serta mengisi ammunition

dan fuel. City juga menyumbang 1000 gold kepada pemain yang memilikinya.

3.2.1.4 Player

Player merepresentasikan pemain yang akan memainkan permainan. *Player* memiliki uang yang dapat digunakan untuk membeli unit. Unit yang telah dibeli digunakan untuk menyerang unit lain atau merebut bangunan. Objek *player* juga nantinya akan digunakan sebagai basis untuk membuat kecerdasan buatan yang akan digunakan dalam aplikasi permainan ini. Attribut utama pada objek *player* adalah:

Tabel 3.8 Attribut Utama Objek Player

Nama	Tipe Data	Keterangan
playerType	Integer	Tipe dari pemain ini. 0 untuk pemain manusia dan 1 untuk komputer
playerSide	Integer	Pihak yang dikendalikan oleh *player* ini pada peta permainan 1 untuk pihak player 1 (sebelah kiri peta permainan) dan 2 untuk pihak player 2 (sebelah kanan peta permainan).
name	String	Nama *player* ini.
***building*s**	Array of *Building*	*Building* yang dimiliki oleh *player* ini.
units	Array of Unit	Unit yang dimiliki oleh *player* ini.
gold	Integer	Jumlah uang yang dimiliki oleh *player* ini. Digunakan untuk membeli unit.

3.2.2 Perancangan Aturan Main Permainan

Aturan main dari aplikasi permainan yang akan dibuat diambil berdasarkan dari "*Advance Wars"* buatan Nintendo.

3.2.2.1 Tahap Persiapan

Pada awal permainan, setiap pemain akan dihadapkan pada sebuah peta permainan (*map*). Pada map tersebut, tiap pemain disediakan sejumlah bangunan (*building*) berupa *HQ*, *Factory*, dan/atau *City* yang jumlahnya sama. Pemain kemudian disediakan sejumlah uang (*gold*) berdasarkan jumlah *building* miliknya dikalikan dengan 1000.

3.2.2.2 Aksi Tiap Giliran

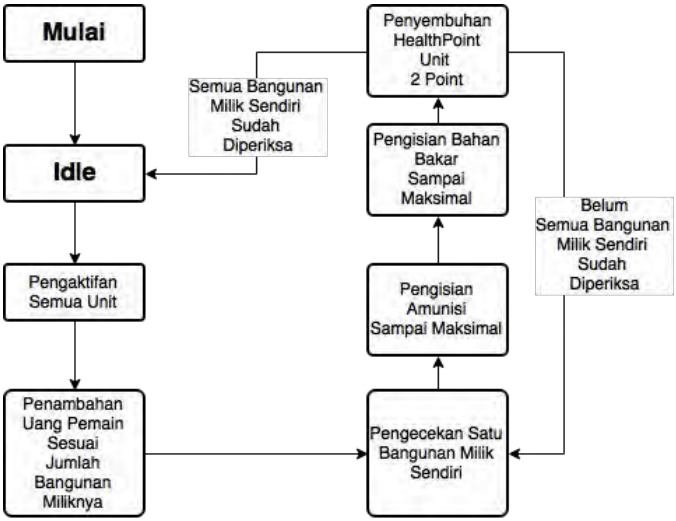
Setelah siap, salah satu pemain akan mengambil giliran pertama. Setelah gilirannya selesai, dilanjutkan pemain yang menjadi lawannya, kemudian kembali ke pemain pertama. Begitu seterusnya sampai tujuan permainan tercapai.

Setiap giliran pemain kemudian melakukan aksi-aksi yang dibagi menjadi beberapa fase: Fase siaga (*standby phase*), Membeli Unit, Menggerakkan Unit, Menyerang Unit dan Mengambilalih bangunan. Fase Siaga selalu dilakukan otomatis setiap awal giliran pemain. Fase lain dapat dilakukan tanpa berurutan setelah fase siaga.

3.2.2.2.1 Fase Siaga (Standby Phase)

Setiap pemain akan memasuki fase siaga pada awal tiap gilirannya. Dalam fase ini, untuk setiap *building* yang dimiliknya, pemain akan mendapatkan *gold* sebesar 1000. Setiap unit akan diatur aktif (*active*) pada fase ini. Setiap unit pemain yang berada di bangunan miliknya akan direparasi (*helathPoint* miliknya akan ditambah) sebesar 2 poin. Amunisi (*ammunition*) dan bahan bakar

(*fuel*) milik unit tersebut juga akan diisi sampai maksimal. Finite state machine (FSM) untuk mekanisme fase siaga ditunjukkan pada Gambar 3.1.

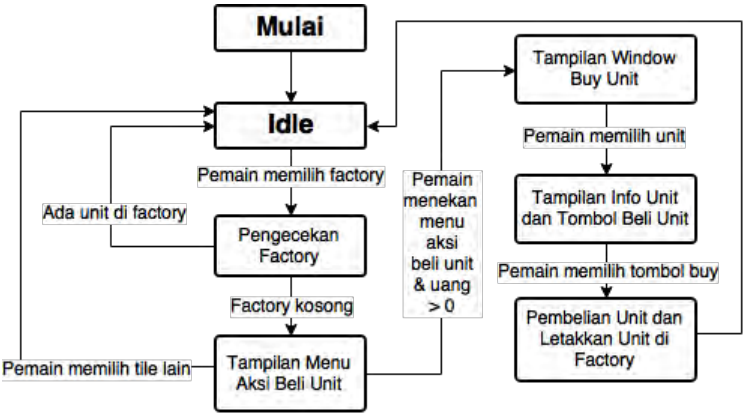


Gambar 3.1 Finite State Machine Mekanisme Fase Siaga

3.2.2.2.2 Membeli Unit

Pemain dapat melakukan pembelian unit melalui bangunan *factory* miliknya, asalkan tidak ada unit (kawan ataupun lawan) yang berada di atasnya.

Pemain dapat melakukan pembelian unit melalui bangunan *factory* miliknya, asalkan tidak ada unit (kawan ataupun lawan) yang berada di atasnya. Untuk membeli unit, pemain harus menyediakan uang sejumlah harga unit yang akan dibeli. Setelah unit terbeli, unit tersebut akan ditempatkan di atas *factory* tempat unit tersebut dibeli, dan akan di-set tidak aktif (*inactive*) yang berarti unit tidak dapat melakukan aksi sampai giliran berikutnya.

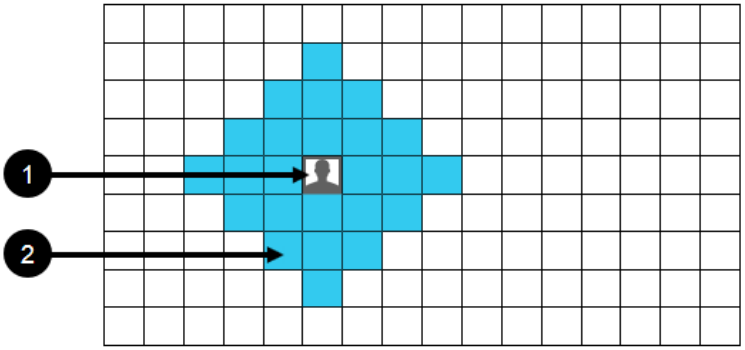


Gambar 3.2 Finite State Machine Membeli Unit

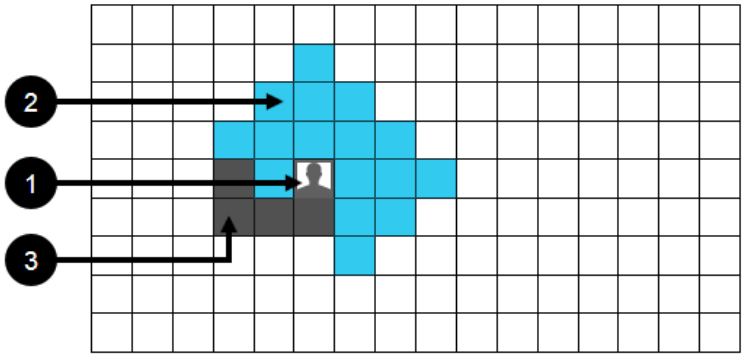
3.2.2.2.3 Memilih Unit

Pemain hanya dapat menggerakkan unit yang statusnya aktif. Sebuah unit harus menyelesaikan gerakannya dahulu sebelum pemain dapat berganti ke unit selanjutnya. Untuk menggerakkan unit, pemain memulainya dengan memilih sebuah unit miliknya yang aktif. Ketika pemain memilih sebuah unit, aplikasi permainan akan menampilkan *tile* mana saja yang dapat dijadikan tujuan gerakan unit tersebut yang disebut *movement tile*. Gambar 3.3 menunjukkan ilustrasi ketika sebuah unit dipilih pemain.

Pada Gambar 3.3, unit yang dipilih oleh pemain ditunjukkan oleh nomor 2. Kumpulan kotak yang berwarna biru dan ditunjukkan oleh nomor 2 adalah *movement tile*. *Movement tile* ini bertujuan untuk membatasi seberapa jauh sebuah unit dapat bergerak. Pada Gambar 3.3 unit tersebuat hanya dapat bergerak maksimal 3 *tile*.



Gambar 3.3 Ilustrasi Pemilihan Unit



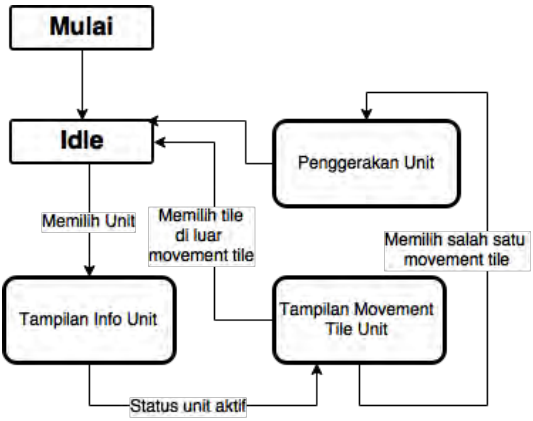
Gambar 3.4 Ilustrasi Pemilihan Unit Dengan Penghalang

Pada Gambar 3.4, gerakan unit terhalang oleh penghalang, yang ditunjukkan nomor 3, di mana unit tersebut tidak dapat melewatinya. Namun, *movement tile* tetap memungkinkan unit tersebut untuk bergerak sampai sejauh maksimal 3 *tile*.

3.2.2.2.4 Menggerakkan Unit

Unit hanya dapat bergerak secara *orthogonal* sesuai empat arah mata angin, dan tidak diperkenankan bergerak secara *diagonal*. Unit tidak dapat bergerak ke *tile* yang sudah ada unit lain diatasnya, baik kawan maupun lawan. Unit dapat bergerak melewati unit kawan, namun tidak dapat melewati unit lawan.

Setiap *terrain* memiliki biaya yang harus dibayar unit ketika melewatinya berdasarkan attribut *movementPoints* yang dimiliki *terrain* di *tile* tersebut.



Gambar 3.5 Finite State Machine Menggerakkan Unit

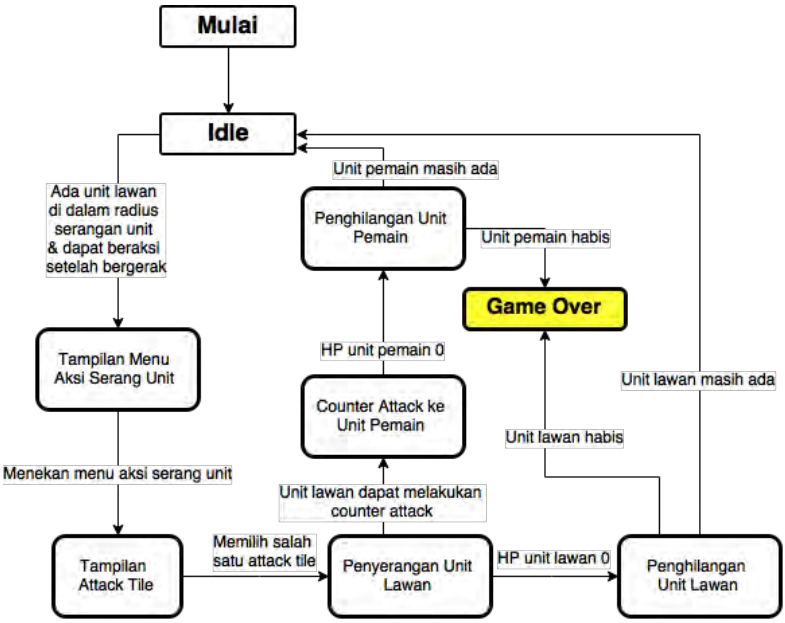
Beberapa unit dapat melewati *terrain* tertentu yang unit lain tidak dapat melewatinya. Contohnya, hanya unit *infantry* dan *mech* yang dapat melewati *terrain* gunung (*mountain*). Hal ini dilihat dari attribut *movementCostOverride* yang dimiliki setiap unit. Beberapa unit ada yang mengonsumsi bahan bakar (*fuel*) ketika bergerak. Attribut *fuel* miliknya akan dikurangi sebesar jumlah *movementCost* terrain yang dilewatinya.

Unit hanya dapat bergerak ke sebuah *tile* yang jaraknya sesuai dengan attribut *movementPoints* dan *fuel* miliknya. Jika nilai attribut *fuel* unit tersebut kurang dari *movementPoints*, maka hanya

menggunakan nilai attribut *fuel*. Jika nilai attribut *fuel* unit tersebut bernilai 0, maka unit tersebut tidak dapat melakukan gerakan.

Jika unit yang berhasil bergerak memiliki attribut "*canActAfterMoving*" maka unit tersebut dapat masuk ke fase menyerang unit lawan ataupun mengambilalih bangunan. Jika tidak maka unit tersebut diatur menjadi *inactive.*

3.2.2.2.5 Menyerang Unit Lawan



Gambar 3.6 Finite State Machine Menyerang Unit Lawan

Jika sebuah unit lawan berada di dalam jangkauan serangan milik sebuah unit pemain (hal ini ditandai dari attribut *minAttackRange* dan *maxAttackRange* yang dimiliki unit tersebut), unit pemain dapat melancarkan serangan ke unit lawan tersebut.

Nilai kerusakan (*damage*) yang dihasilkan dari serangan tersebut diambil dari persamaan 3.1.

$$D = \frac{bDmg * \left(100 - \left((dCov * dHp)\right)\right) * \left(\frac{aHp}{10}\right) * (100)}{10000} \quad (3.1)$$

Di mana:

D : *Damage*, besar nilai kerusakan dari serangan, dalam persen.
bDmg : *base damage*, diambil dari attribut *primaryDamage* milik unit penyerang, jika nilainya 0, maka diambil dari attribut *secondaryDamage*.
dCov : *defender cover*, nilai dari attribut *cover* milik *terrain* di mana unit yang diserang berada.
dHp : *defender HP*, nilai dari attribut *healthPoint* milik unit yang diserang.
aHp : *defender HP*, nilai dari attribut *healthPoint* milik unit yang menyerang.

Nilai hasil persamaan tersebut kemudian dibulatkan ke bawah. Attribut *healthPoint* (HP) milik unit lawan kemudian dikurangi sebesar nilai yang dihasilkan dengan persamaan berikut:

$$R = D / 10 \quad (3.2)$$

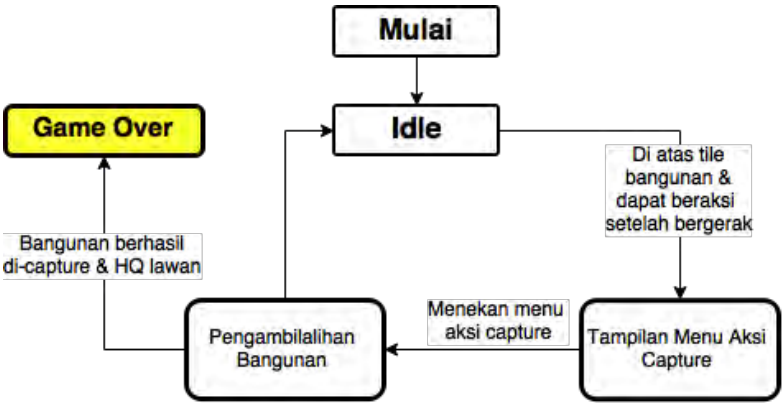
Di mana:

R : *Reduction*, besar nilai untuk mengurangi HP milik unit yang diserang.
D : *Damage*, besar nilai kerusakan dari serangan, dalam persen.

Untuk kemudian nilai *reduction* tersebut dibulatkan ke bawah atau ke atas secara *random*. Jika unit yang melakukan serangan menggunakan *primaryDamage* pada saat melakukan serangan, maka attribut *ammunition* miliknya dikurangi sebesar 1, namun jika menggunakan *secondaryDamage*, maka tidak dikurangi apapun. Unit yang diserang dapat melancarkan serangan balasan (*counter attack*) jika attribut "*canCounter*" miliknya bernilai *true*. Besar *damage* dan *reduction* yang dihasilkan menggunakan persamaan yang sama dengan persamaan serangan biasa. Jika setelah dikurangi nilai *healthPoint* milik unit yang diserang bernilai 0 atau kurang, maka unit tersebut dihilangkan dari permainan. Unit yang melakukan serangan, jika masih bertahan hidup, kemudian diatur menjadi *inactive*.

3.2.2.2.6 Mengambilalih Bangunan

Unit dapat mengambilalih (*capture*) bangunan (*building*) netral atapun milik lawan jika attribut "*canCapture*" milik unit tersebut bernilai "*true*". Untuk melakukan *capture*, unit pemain harus bergerak ke *tile* di mana *building* tersebut berada.



Gambar 3.7 Finite State Machine Mengambilalih Bangunan

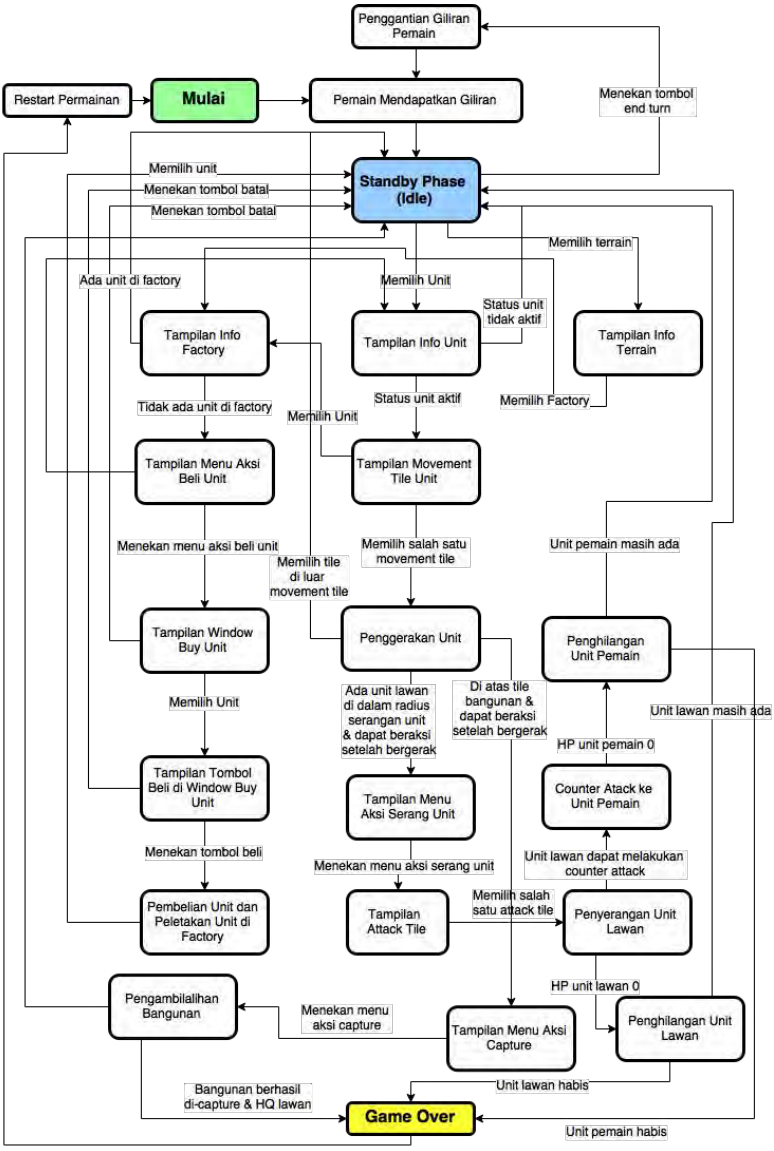
Attribut *capturePoints* milik *Building* tersebut kemudian dikurangi sebesar nilai attribut *healthPoint* milik unit yang melakukan *capture*. Jika setelah melakukan *capture*, di giliran berikutnya, sebuah unit bergerak meninggalkan *building* yang sedang dilakukan *capture*, maka attribut *capturePoints* milik *building* tersebut dikembalikan ke nilai awalnya, yang diambil dari attribut *maxCapturePoints*. Jika setelah dilakukan *capture* oleh pemain, nilainya attribut *capturePoints* milik sebuah bangunan menjadi 0 atau kurang, maka *building* tersebut menjadi milik pemain. Unit yang telah melakukan *capture* langsung di-*set inactive*.

3.2.2.2.7 Pengecekan Akhir Permainan

Pemain dinyatakan menang apabila berhasil menghabisi semua unit atau mengambil alih bangunan "*HQ*" milik lawannya.

3.2.2.3 Finite state machine Permainan

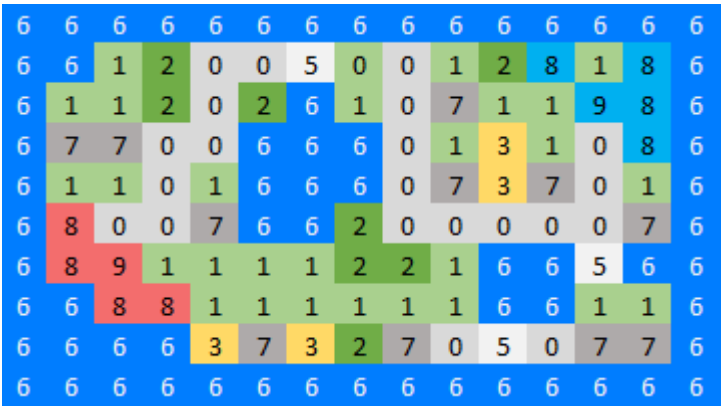
Berdasarkan rancangan aturan permainan yang telah dibuat, diagram *Finite State Machie* (*FSM*) dari seluruh aturan permainan ini adalah seperti yang ditunjukkan Gambar 3.8 berikut.



Gambar 3.8 Finite State Machine Permainan

3.2.3 Perancangan Peta Permainan

Hanya ada satu peta permainan yang digunakan dalam apliaksi permainan ini. Peta permainan tersebut berbentuk *grid* atau *array* dua dimensi yang diambil dari peta asli yang digunakan di permainan Advance Wars bernama "*Spann Island*". Peta permainan ini berukuran 15 x 10 *tile* dengan desain sebagai berikut:



Gambar 3.9 Desain Peta Permainan

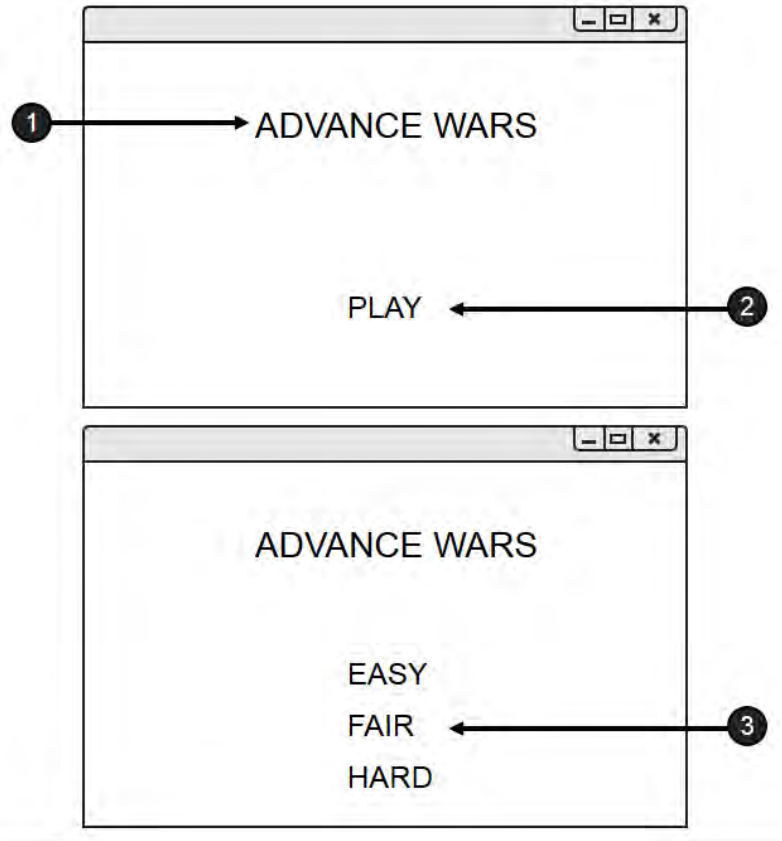
Pada Gambar 3.9 terdapat angka-angka yang menunjukkan attribut *terrainID* untuk setiap *tile* yang bersangkutan.

3.2.4 Perancangan Antarmuka Permainan

Tahap perancangan antarmuka dalam subbab ini membahas perancangan antarmuka dari permainan. Perancangan antarmuka ini bertujuan untuk memberikan gambaran proses pengembangan mengenai tampilan antarmuka aplikasi permainan ini.

3.2.4.1 Perancangan Antarmuka Layar Main Menu

Layar *main menu* adalah tampilan yang pertama kali dilihat pemain. Di layar ini pemain dapat memilih tingkat kesulitan komputer yang akan dilawan. Tampilan layar *main menu* adalah sebagai berikut:



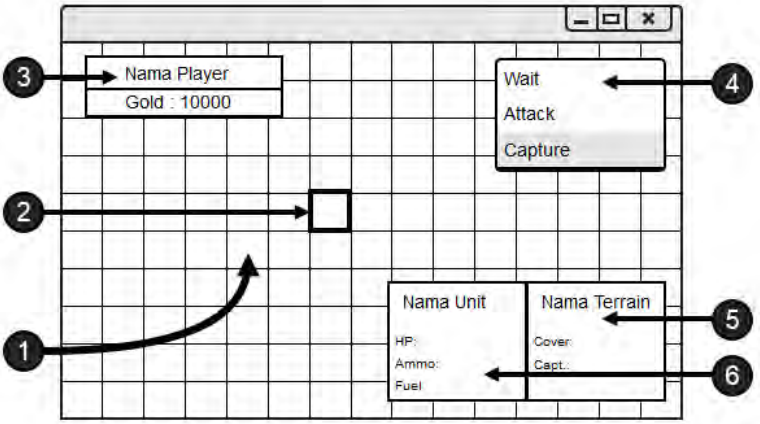
Gambar 3.10 Tampilan Antarmuka Layar *Main Menu*

Pada Gambar 3.10 terdapat komponen-komponen dari antarmuka layar *main menu* sebagai berikut:

1. Logo aplikasi permainan.
2. Tombol *Play*. Ketika ditekan akan berganti *menjadi list menu difficulty*.
3. *List Menu Difficulty*. Tingkat kesulitan komputer yang dapat dipilih pemain.

3.2.4.2 Perancangan Antarmuka Layar Battle

Layar *battle* adalah layar utama dalam permainan ini, di mana pemain akan bertarung melawan pemain lain. Dalam layar inilah pemain dapat membeli unit, menggerakkan unit, menyerang unit lawan, merebut bangunan, dan aksi lain dalam peraturan permainan. Tampilan dari layar *battle* adalah sebagai berikut:



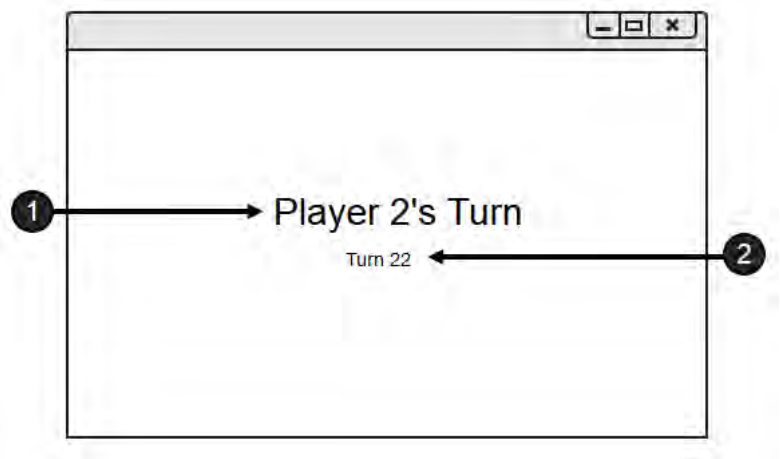
Gambar 3.11 Tampilan Antarmuka Layar *Battle*

Dari Gambar 3.11, komponen-komponen yang terdapat pada layar *battle* adalah sebagai berikut:

1. *Battle Map*, atau peta permainan, di mana objek-objek permainan, seperti *unit*, *terrain*, dan *building* akan diletakkan
2. *Cursor*, sebagai penanda objek permainan yang dipilih pemain.
3. *Player Info*. Menampilkan informasi nama pemain dan jumlah uang (*gold*) yang dimilikinya.
4. *Action Menu*. Berisi aksi-aksi yang dapat dilakukan pemain. Isi dari menu ini akan berubah tergantung dari fase pemain tersebut, apakah sedang *idle*, menggerakkan unit, dan sebagainya.
5. *Terrain Info*. Muncul jika pemain memilih *terrain* pada *battle map*. Menampilkan informasi *terrain* yang dipilih oleh pemain, berisi nama, nilai attribut *cover*, dan nilai attribut *capture points* milik *terrain* tersebut.
6. *Unit Info*. Muncul jika pemain memilih unit pada *battle map*. Menampilkan informasi unit, berisi nama, nilai attribut *healthPoint*, nilai attribut *ammunition*, dan nilai attribut *fuel* dari unit tersebut.

3.2.4.3 Perancangan Antarmuka Layar Change Turn

Layar ini akan muncul saat seorang pemain mengakhiri gilirannya dan berganti ke pemain lawannya, yaitu dengan memilih menu aksi "*end turn"*. Tampilan dari layar *change turn* adalah sebagai berikut:



Gambar 3.12 Tampilan Antarmuka Layar *Change Turn*

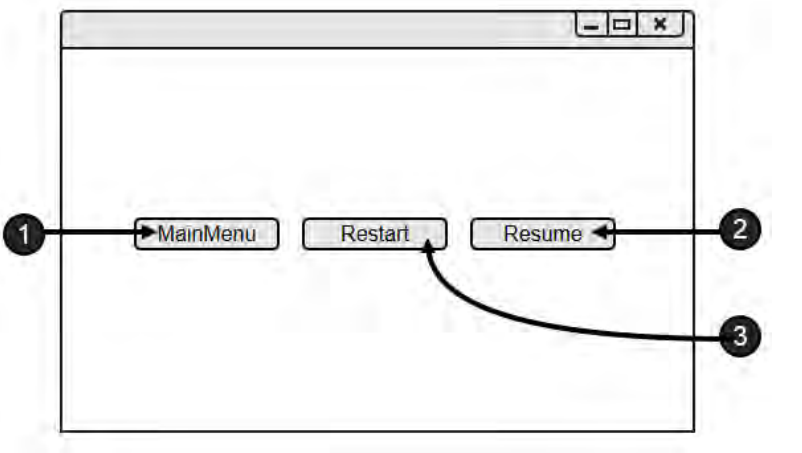
Komponen-komponen tampilan pada antarmuka layar *change turn* dari Gambar 3.12 adalah sebagai berikut:

3.

1. *Label Player Name*. Menampilkan nama player yang mendapat giliran
2. *Label Total Turn*. Menampilkan jumlah giliran yang sudah dilewati.

3.2.4.4 Perancangan Antarmuka Layar Pause

Layar *pause* akan muncul jika pemain memilih aksi *pause* pada layar *battle*. Pada layar ini pemain dapat memilih untuk melanjutkan kembali permainan, mengulang permainan dengan tingkat kesulitan yang sama, atau keluar ke layar *main menu*. Tampilan dari layar *pause* adalah sebagai berikut:



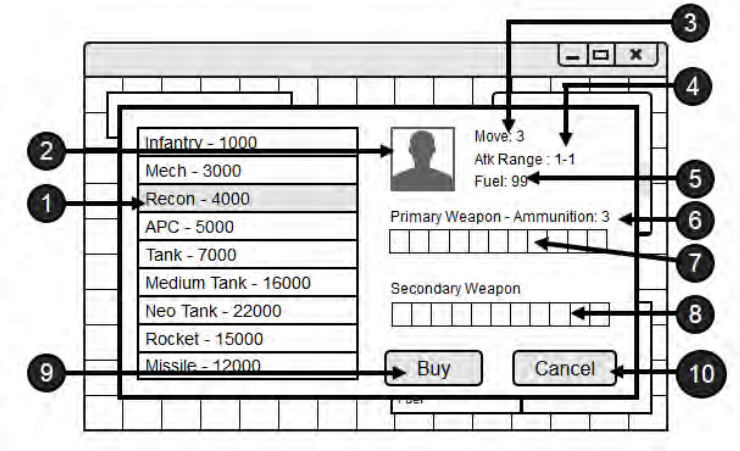
Gambar 3.13 Tampilan Antarmuka Layar *Pause*

Komponen-komponen di dalam layar *pause* pada gambar 3.13 adalah sebagai berikut:

1. Tombol *Main Menu*. Berfungsi mengganti layar ke layar *main menu*.
2. Tombol *Restart*. Berfungsi mengulang permainan dengan tingkat kesulitan yang sama.
3. Tombol *Resume*. Berfungsi untuk kembali ke layar *battle*.

3.2.4.5 Perancangan Antarmuka Window Buy Unit

Window atau jendela ini akan muncul jika pemain memilih aksi "*buy unit*" pada layar battle. Tampilan dari *window buy unit* adalah sebagai berikut:



Gambar 3.14 Tampilan Antarmuka *Window Buy Unit*

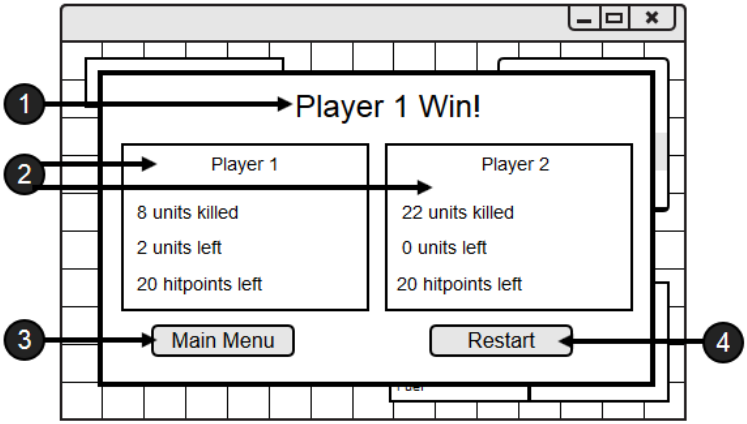
Komponen-komponen yang ada di dalam *window buy unit* pada Gambar 3.14 adalah sebagai berikut:

1. *List Menu Unit*. Berisi nama-nama unit apa saja yang dapat dibeli pemain berdasarkan uang yang dimilikinya.
2. *Unit Preview*. Menampilkan gambar dari unit yang dipilih pemain.
3. *Label Move*. Menampilkan nilai attribut *move* milik unit yang dipilih
4. *Label Attack Range*. Menampilkan nilai attribut *minAttackRange* dan *maxAttackRange* milik unit yang dipilih
5. *Label Fuel*. Menampilkan nilai attribut *fuel* milik unit yang dipilih.
6. *Label Ammunition*. Menampilkan nilai attribut *ammunition* milik unit yang dipilih.
7. *List Primary Target*. Menampilkan daftar ikon unit yang dapat diserang unit yang dipilih menggunakan *primary damage*.

8. *List Secondary Target*. Menampilkan daftar ikon unit yang dapat diserang unit yang dipilih menggunakan *secondary damage*.
9. *Tombol Buy*. Berfungis untuk mengonfirmasi pembelian unit yang dipilih.
10. *Tombol Cancel*. Berfungi untuk menutup *window buy unit*.

3.2.4.6 Perancangan Antarmuka Window Game Over

Window game over akan muncul ketika salah satu pemain berhasil mengalahkan lawannya. Tampilan dari *window game over* adalah sebagai berikut:



Gambar 3.15 Tampilan Antarmuka *Window Game Over*

Komponen-komponen antarmuka milik *window game over* yang terdapat pada Gambar 3.15 adalah sebagai berikut:

1. *Label Player Name*. Menampilkan nama pemain yang memenangkan permainan.

2. *Panel Info Battle*. Menampilkan data hasil permainan untuk tiap pemain, yang berisi: jumlah unit terbunuh, jumlah unit tersisa, dan jumlah *hitPoints* yang tersisa.
3. *Tombol Main Menu*. Berfungsi untuk kembali ke *Layar Select Player 1*.
4. *Tombol Restart*. Berfungsi untuk mengulang permainan dengan pemain untuk *player 1* dan *player 2* sama seperti permainan sebelumnya.

3.2.5 Perancangan Kecerdasan Buatan

Perancangan kecerdasan buatan yang dibahas pada subbab ini dibagi menjadi dua, yaitu perancangan untuk kecerdasan buatan adaptif dan untuk kecerdasan buatan lawan.

Perancangan kecerdasan buatan adaptif akan membahas perancangan kecerdasan buatan utama yang menjadi pokok bahasan tugas akhir penulis, sedangkan perancangan kecerdasan buatan akan membahas perancangan kecerdasan buatan yang nantinnya akan menjadi lawan dari kecerdasan buatan adaptif pada tahap pengujian.

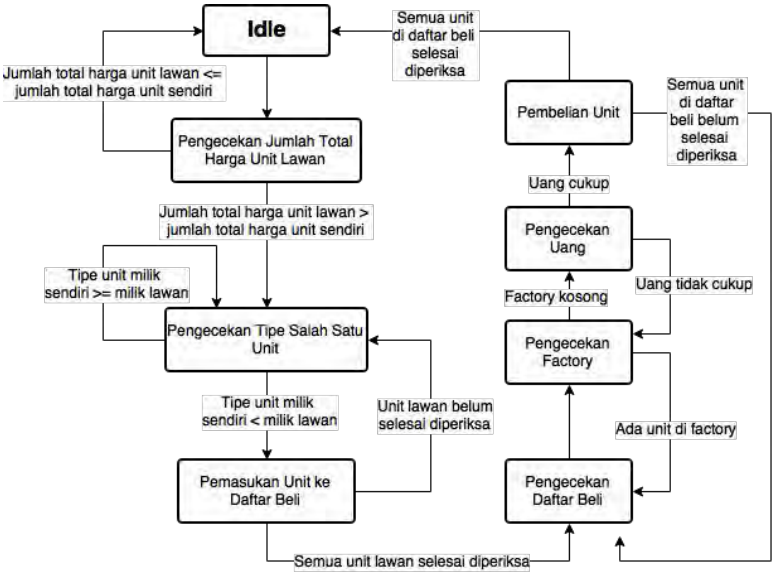
3.2.5.1 Perancangan Kecerdasan Buatan Adaptif

Kecerdasan buatan adaptif (untuk selanjutnya akan disebut *AI_Adaptive*) yang akan digunakan dalam aplikasi permainan ini nantinya akan menggunakan fungsi evaluasi dan beberapa mekanisme untuk dapat menyeimbangi permainan lawannya. Mekanisme-mekanisme tersebut yaitu digunakan pada saat pembelian unit, penyerangan unt lawan, dan pengambilalihan bangunan.

3.2.5.1.1 Pembelian Unit

Pada saat *AI_Adaptive* mendapat giliran, hal pertama yang dilakukannya adalah membeli unit yang sesuai dengan keadaan

lawannya. *Finite state machine (FSM)* mekanisme pembelian unit pada *AI_Adaptive* ditunjukkan pada Gambar 3.16.



Gambar 3.16 FSM Mekanisme Pembelian Unit AI_Adaptive

Pertama *AI_Adaptive* akan memeriksa, apakah lawannya memiliki unit. Jika lawan tidak punya unit sama sekali, maka *AI_Adaptive* akan langsung akan mengakhiri gilirannya. Jika lawan memiliki unit dan jumlah nilai harganya lebih besar dari jumlah unit miliknya, *AI_Adaptive* akan melihat setiap tipe unit yang dimiliki lawannya, kemudian membandingkan dengan yang dimiliknya, jika lawannya memiliki tipe unit tersebut lebih banyak dari dirinya, maka ia akan mencatatnya pada daftar unit yang akan dibeli sebesar selisihnya. Daftar tersebut kemudian akan diurutkan berdasarkan besar harganya. Unit dengan harga tertinggi akan berada di urutan teratas.

Setelah daftar beli selesai dibuat, *AI_Adaptive* kemudian akan memeriksa *factory* miliknya secara random. Untuk setiap

factory yang kosong, *AI_Adaptive* kemudian akan membeli unit pada daftar pesanan yang paling atas jika uangnya mencukupi.

3.2.5.1.2 Pergerakan dan Aksi Unit

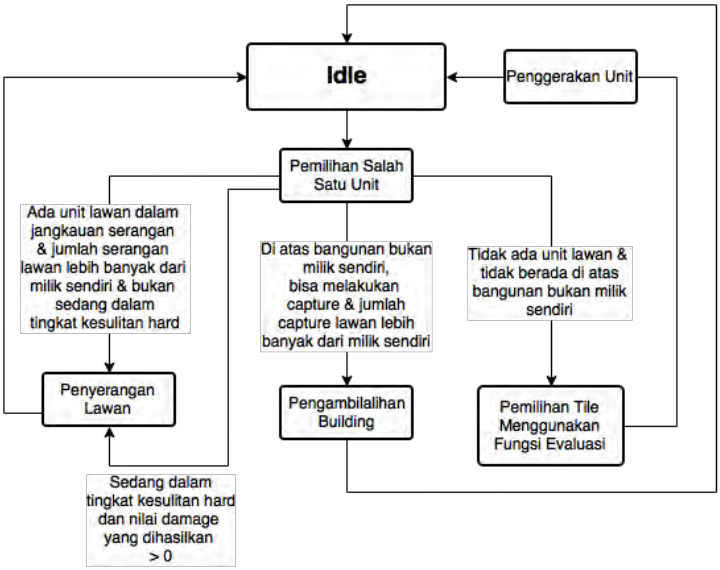
Setelah pembelian unit selesai dilakukan, *AI_Adaptive* akan melakukan aksi terhadap unitnya satu persatu.

Pertama unit tersebut kemudian akan memeriksa sekitarnya apakah ada lawan yang berada dalam radius serangannya. Jika ada, *AI_Adaptive* akan membandingkan jumlah serangan yang telah dilakukannya dari awal permainan dengan milik pemain lawan, jika lebih kecil dan bukan dalam tingkat kesulitan *hard*, maka unit tersebut akan melakukan serangan ke unit lawan yang berada di jarak serangannya. Jika tingkat kesulitan yang dipilih pemain adalah hard, maka *AI_Adaptive* akan langsung menyerang unit lawan tanpa memeriksa jumlah serangan terlebih dulu.

Jika unit tersebut berada di atas sebuah bangunan netral atau lawan dan bisa mengambil alih (*capture*), *AI_Adaptive* juga akan membandingkan jumlah *capture* miliknya dengan lawan yang telah dilakukan dari awal permainan. Unit hanya akan melakukan *capture building* di posisinya jika nilai jumlah *capture*-nya lebih kecil.

Jika tidak ada lawan dan tidak sedang berada di atas sebuah bangunan, maka *AI_Adaptive* akan menggerakkan unit-unitnya. Untuk setiap unit tersebut, *AI_Adaptive* akan menghitung skor pada setiap *tile* di mana unit tersebut dapat bergerak. Skor tersebut didapatkan menggunakan fungsi evaluasi. Skor kemudian dibandingkan dengan skor milik lawan pada giliran sebelumnya, lalu dibandingkan dengan tingkat kesulitan yang dipilih pemain lawan pada saat sebelum memulai permainan.

Finite state machine (FSM) untuk mekanisme pergerakan unit ditunjukkan pada Gambar 3.17.



Gambar 3.17 FSM Mekanisme Pergerakan Unit AI_Adaptive

3.2.5.1.3 Fungsi Evaluasi

Fungsi evaluasi digunakan untuk mengevaluasi seberapa bagus sebuah *tile* yang menjadi tujuan gerakan unit pada saat akan bergerak. Fungsinya ditunjukkan pada persamaan 3.3[15].

$$S(x,y) = \sum_{c \in C_E} \beta\big(EN, DS, \delta(c,x,y)\big) + \sum_{c \in C_A} \gamma\big(EN, \delta(c,x,y)\big) \quad (3.3)$$

Di mana :

C_A merepresentasikan set dari semua unit kawan.

C_E merepresentasikan set dari semua unit lawan.
β(EN,DS,δ): menghitung *enemy partial score* menggunakan :
EN: Informasi lingkungan lawan (*Enemies environment information*)
DS: Skor kerusakan (*Damage Score*).
δ : Jarak dari unit lawan.
γ(EN, δ): menghitung *ally partial score* menggunakan :
EN: Informasi lingkungan kawan (*Allies environment information).*
δ : Jarak dari unit yang akan bergerak.
δ(c,x,y): menghitung jarak unit c ke *tile* (x,y).

Fungsi evaluasi di atas digunakan untuk menghitung prediksi skor yang dihasilkan jika sebuah unit ke sebuah *tile*(x,y) dengan menggunakan informasi yang diperoleh dari set kawan (*ally*) dan lawan (*enemy*) serta jarak unit tersebut ke *tile*(x,y). Informasi tersebut berisi keadaan *enemy* atau *ally* saat unit tersebutakan bergerak. Dalam fungsi evaluasi di atas terdapat dua fungsi utama untuk menghasilkan skor parsial, yaitu fungsi untuk kalkulasi terhadap semua unit lawan (*enemy function*) dan fungsi untuk kalkulasi terhadap unit kawan (*ally function*).

Enemy function merupakan bagian dari fungsi evaluasi yang berguna untuk melakukan kalkulasi *enemy partical score* terhadap setiap unit lawan jika sebuah unit bergerak ke sebuah *tile*. Fungsinya ditunjukkan pada persamaan 3.4.

$$\sum_{c \in C_E} \beta\big(EN, DS, \delta(c,x,y)\big) \quad (3.4)$$

Rumus untuk menghasilkan *enemy partial score* di setiap *tile* ditunjukkan pada persamaan 3.5.

$$\begin{aligned}\beta = \quad & (1 + (cov * 10)) \\ & * (1 + (\delta * 10)) \\ & * (1 + (\Delta stat * 10)) \\ & * (1 + (DS * 10)) \\ & * (1 + (\Delta hp * 10)) \\ & / (1 + (d_s * 10))\end{aligned} \tag{3.5}$$

Keterangan:

β : *enemy partial score*. Skor parsial terhadap set *enemy*.
cov : *cover*. Nilai *cover* untuk *terrain* pada *tile*(x,y).
δ : *distance*. Jarak unit dengan *tile*(x,y).
Δstat : *delta stats*. Nilai perbedaan *statistik* sebuah unit dengan unit lawan ini.
DS : *damage score*. Nilai perkiraan kerusakan yang dihasilkan unit sebuah jika menyerang unit lawan ini di *tile* (x,y).
Δhp : *delta hitPoints*. Nilai perbedaan attribut *hitPoints* sebuah unit dengan unit lawan ini.
d_s : *enemy distance*. Jarak unit lawan ini dengan *tile*(x,y).

Variabel *cov,* Δ*stat,* Δ*hp* merupakan bagian dari *Enemies environment information* dikarenakan variabel-variabel tersebut berpengaruh terhadap bagus tidaknya sebuah *tile* terhadap unit lawan. Variabel *cov,* Δ*stat,* dan Δ*hp* berpengaruh pada saat pertarungan dua unit, semakin besar nilainya semakin besar kemungkinan untuk menang saat pertarungan.

Ally function merupakan bagian dari fungsi evaluasi yang digunakan untuk melakukan kalkulasi *ally partial score* terhadap setiap unit kawan jika sebuah unit bergerak ke sebuah *tile*. Fungsinya ditunjukkan pada persamaan 3.6.

$$\sum_{c \in C_A} \gamma\big(EN, \delta(c, x, y)\big) \tag{3.6}$$

Rumus untuk menghasilkan *ally partial score* di setiap *tile* ditunjukkan pada persamaan 3.7.

$$\gamma = (1 + (hp_a * 10)) / (1 + (d_a * 10)) * (1 + (\delta * 10)) \quad (3.7)$$

Keterangan:

γ : *ally partial score*. Skor parsial terhadap set ally.
hp_a : *ally hitPoints*. Nilai attribut *hitPoints* milik unit *ally* ini.
d_a : *ally distance*. Jarak unit *ally* ini dengan *tile*(x,y).
δ : *distance*. Jarak unit dengan *tile*(x,y).

Variabel d_a dan hp_a merupakan bagian dari *Allies environment information* dikarenakan variabel-variabel tersebut berpengaruh terhadap bagus tidaknya sebuah *tile* terhadap unit kawan. Variabel d_a,dan hp_a berpengaruh karena semakin dekat sebuah unit terhadap unit kawan, maka besar kemungkinan unit kawan tersebut akan membantu pada saat pertarungan dengan lawan.

3.2.5.1.4 Pengaturan Tingkat Kesulitan

Setelah menghitung nilai yang didapat dari fungsi evaluasi, *AI_Adaptive* kemudian akan menghitung skor untuk setiap *tile* untuk menilai seberapa imbang sebuah *tile* jika unit bergerak ke situ menggunakan persamaan 3.8[15]:

$$E(x,y) = S_{AI}(x,y) - \frac{\sum_{t \in T} S_{OP}}{N} \quad (3.8)$$

Keterangan:

E(x,y) : tingkat keseimbangan skor untuk setiap *tile* (x,y).
S_{AI}(x,y) : skor kecerdasan buatan, jika bergerak ke *tile* (x,y).
S_{OP} : skor dari seorang pemain manusia yang bergerak setelah giliran kecerdasan buatan sebelumnya. Skor ini perlu dijumahkan dikarenakan, ada kemungkinan untuk dua atau lebih unit dari pihak yang sama untuk bergerak dalam satu giliran.
T : set dari giliran pemain manusia.
N : jumlah anggota di dalam T.

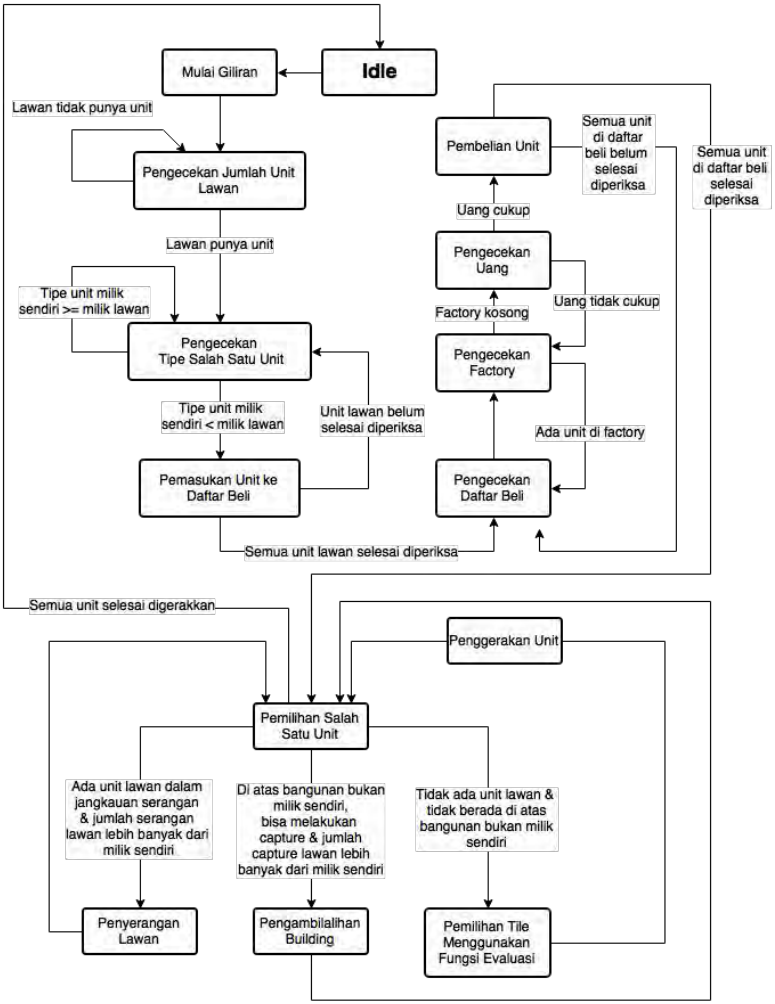
Nilai E positif berarti *AI_Adaptive* berada dalam posisi yang lebih baik dibandingkan lawannya, dan sebaliknya. Jika nilai E sama dengan nol, berarti kedua belah pihak melakukan gerakan secara seimbang dalam giliran tersebut.

AI_Adaptive kemudian akan memilih satu *tile* yang akan dijadikan tujuan gerakan. Pemilihan *tile* ini berdasarkan tingkat kesulitan yang dipilih oleh pemain sebelum permainan dimulai.

Jika pemain memilih tingkat kesuliatan mudah atau *easy*, *AI_Adaptive* akan memilih *tile* dengan skor paling rendah. Sebaliknya jika pemain memilih tingkat kesulitan susah atau *hard*, *AI_Adaptive* akan memilih *tile* dengan skor paling tinggi. Sedangkan jika pemain memilih tingkat kesulitan imbang atau *fair*, *AI_Adaptive* akan memilih *tile* dengan skor yang sedekat mungkin dengan nol.

3.2.5.1.5 Finite state machine Kecerdasan Buatan Adaptif

Berdasarkan rancangan mekanisme pembelian, pergerakan, dan aksi unit, *finite state machine* dari *AI_Adaptive* ditunjukkan pada Gambar 3.18.



Gambar 3.18 FSM AI_Adaptive

3.2.5.2 Perancangan Kecerdasan Buatan Lawan

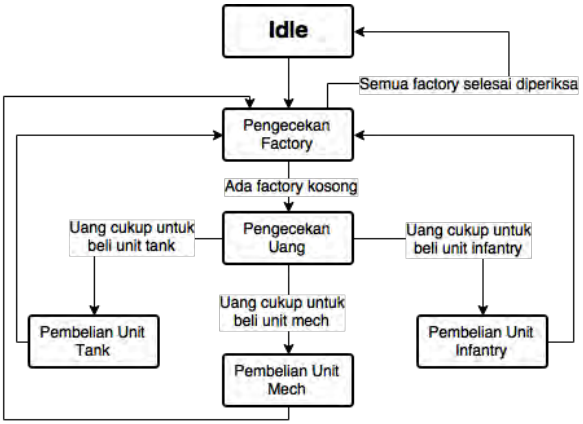
Kecerdasan buatan lawan yang akan dipakai dalam permainan ini dipakai pada saat proses pengujian. Perancangan kecerdasan buatan lawan ini berdasarkan beberapa strategi bermain permainan *turn-based strategy* yang telah dibahas pada bab sebelumnya. Kecerdaan lawan yang akan dibangun adalah *AI_Rush, AI_UnitOffence, AI_Defense*, dan *AI_Balance*.

3.2.5.2.1 Pembelian Unit

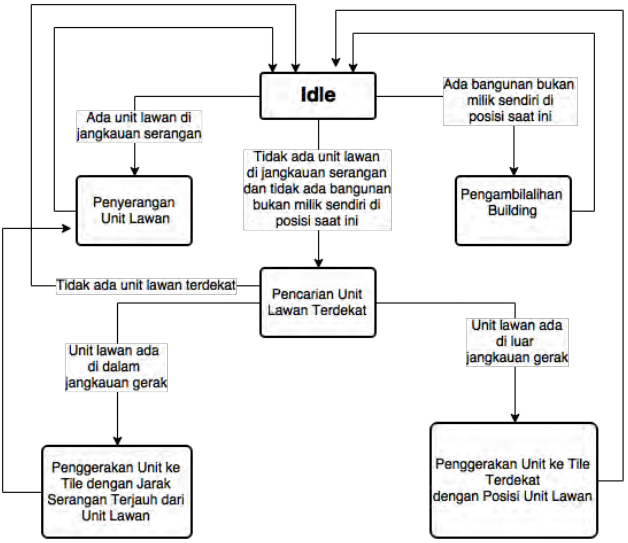
Semua kecerdasan buatan lawan yang digunakan dalam permainan ini memiliki mekanisme pembelian unit yang sama. Pertama kecerdasan buatan akan memeriksa setiap *factory* yang dimilikinya apakah kosong atau tidak. Jika kosong, langkah selanjutnya adalah membeli unit sesuai urutan prioritas jika uang yang dimiliki mencukupi. Prioritas tertinggi adalah memebeli unit tank. Jika uangnya tidak cukup, maka kecerdasan buatan lawan akan membeli unit mech. Jika tidak cukup juga, maka kecerdasan buatan lawan akan membeli unit infantry. *Finite state machine* (*FSM*) mekanisme pembelian unit pada kecerdasan buatan lawan ditunjukkan pada Gambar 3.19.

3.2.5.2.2 Pergerakan dan Aksi Unit Pada AI_Rush

AI_Rush adalah kecerdasan buatan yang dirancang berdasarkan strategi *Rush*. Pergerakan pada *AI_Rush* berfokus pada penyerangan unit lawan yang terdekat. *Finite state machine* untuk *AI_Rush* ditunjukkan pada Gambar 3.20.



Gambar 3.19 FSM Mekanisme Pembelian Unit Kecerdasan Buatan Lawan



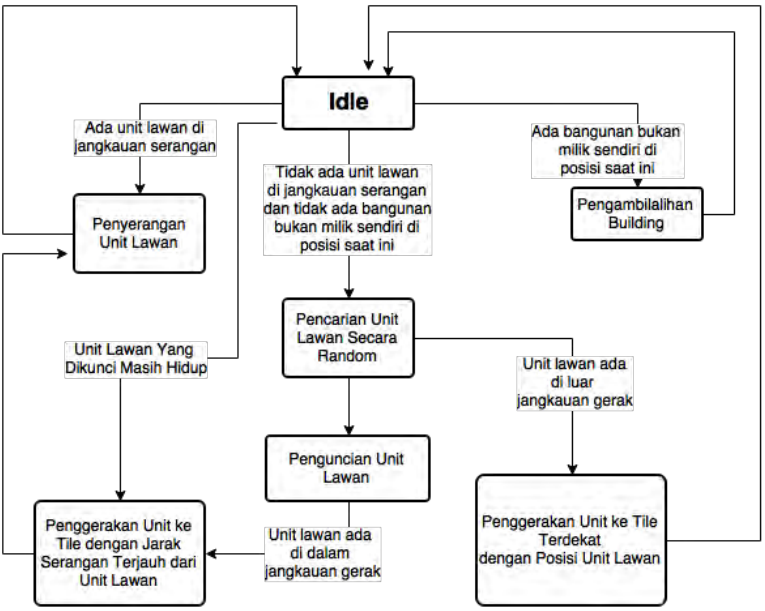
Gambar 3.20 FSM Mekanisme Pergerakan Unit AI_Rush

Pertama *AI_Rush* akan mencari unit lawan yang ada di jangkauan serangan. Jika ada, ia akan menyerang unit tersebut. Jika tidak, ia akan memeriksa apakah unit yang sedang digerakkan sedang berada di atas bangunan yang bukan miliknya. Jika ya, unit akan melakukan *capture* pada bangunan itu. Jika tidak, unit tersebut akan mencari unit lawan dengan jarak terdekat. Jika unit lawan tersebut berada di jarak serangan, maka unit akan bergerak ke *tile* dengan jarak serangan terjauh dari unit lawan tersebut, kemudian menyerangnya. Jika unit lawan berada di luar jangkauan gerak, unit akan bergerak ke posisi terdekat dengan lawan.

3.2.5.2.3 Pergerakan dan Aksi Unit Pada AI_UnitOffence

AI_UnitOffence adalah kecerdasan buatan yang dirancang berdasarkan strategi *UnitOffence*. Pergerakan unit pada *AI_UnitOffence* berfokus pada penguncian sebuah unit lawan untuk kemudian diserang sampai unit tersebut kalah, sebelum berpindah ke sasaran lain. *Finite state machine* untuk *AI_UnitOffence* ditunjukkan pada Gambar 3.21.

Pertama *AI_UnitOffence* akan akan mencari unit lawan yang ada di jangkauan serangan. Jika ada, ia akan menyerang unit tersebut. Jika tidak, ia akan memeriksa apakah unit yang sedang digerakkan sedang berada di atas bangunan yang bukan miliknya. Jika ya, unit akan melakukan *capture* pada bangunan itu. Jika tidak, *AI_UnitOffence* akan memeriksa apakah unit tersebut sedang mengunci sebuah unit lawan, jika ya dan unit lawan yang dikunci masih hidup, unit akan menyerang unit lawan. Jika unit lawan yang dikunci tersebut sudah kalah atau unit tersebut sedang tidak mengunci unit lawan, unit akan mencari target lagi secara random kemudian menguncinya.



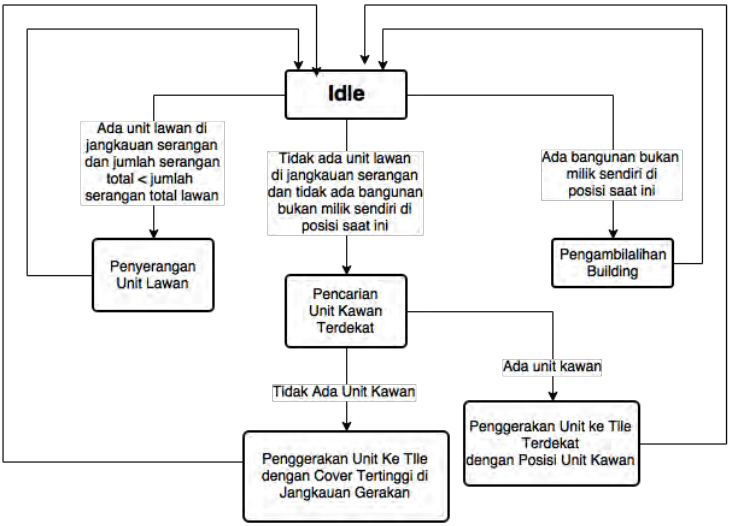
Gambar 3.21 FSM Mekanisme Pergerakan Unit AI_UnitOffence

3.2.5.2.4 Pergerakan dan Aksi Unit Pada AI_Defensive

AI_Defensive adalah kecerdasan buatan yang dirancang berdasarkan strategi *Defensive*. Pergerakan unit pada *AI_Defensive* berfokus pada strategi yang menjaga setiap unit kawan saling berdekatan saru-sama lain, dan hanya menyerang jika diserang lebih dulu. *Finite state machine* untuk *AI_Defensive* ditunjukkan pada Gambar 3.22.

Pertama *AI_Defensive* akan akan mencari unit lawan yang ada di jangkauan serangan. Jika ada dan jumlah total serangan lawan lawan lebih besar dari jumlah serangan miliknya, ia akan menyerang unit tersebut. Jika tidak, ia akan memeriksa apakah unit yang sedang digerakkan sedang berada di atas bangunan yang bukan miliknya. Jika ya, unit akan melakukan *capture* pada

bangunan itu. Jika tidak, *AI_Defensive* akan memeriksa apakah ada unit kawan terdekat, jika ada unit akan bergerak ke *tile* terdekat dengan unit kawan tersebut, jika tidak ada unit akan bergerak ke *tile* dengan nilai attribut *cover* tertinggi di jangkauan gerakan.

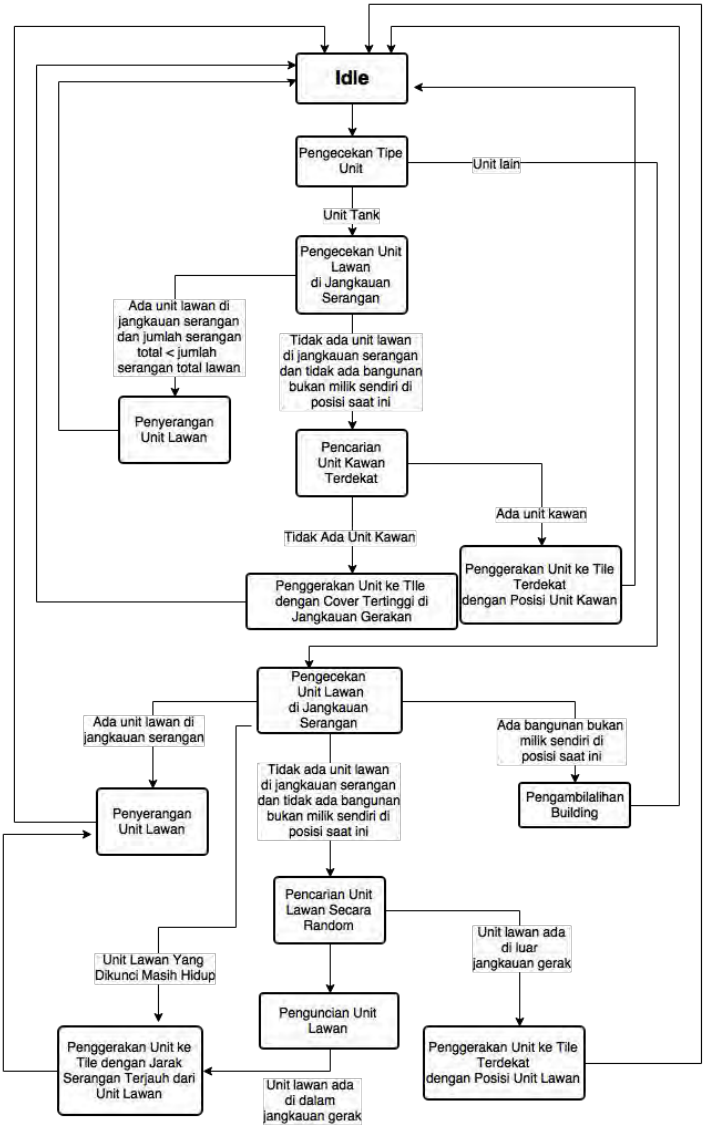


Gambar 3.22 FSM Mekanisme Pergerakan Unit AI_Defensive

3.2.5.2.5 Pergerakan dan Aksi Unit Pada AI_Balance

AI_Balance adalah kecerdasan buatan yang dirancang berdasarkan strategi Balance. Pergerakan unit pada *AI_Balance* berfokus pada campuran mkeanisme pergerakan unit di *AI_Defensive* dan *AI_UnitOffence*. *Finite state machine* untuk *AI_Balance* ditunjukkan pada Gambar 3.23.

Pertama, *AI_Balance* akan memeriksa tipe unit. Jika tipenya adalah unit *"tank"*, maka ia akan bertindak seperti *AI_UnitOffence*, sedangkan Jika tipenya adalah unit yang lain ia akan bertindak seperti *AI_Defensive*.



Gambar 3.23 FSM Mekanisme Pergerakan Unit AI_Balance

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI SISTEM

Bab ini membahas tentang implementasi dari perancangan sistem. Bab ini berisi proses implementasi dari setiap kelas pada semua modul. Namun, pada hasil akhir mungkin saja terjadi perubahan kecil. Bahasa pemrograman yang digunakan adalah bahasa pemrograman C++ dengan tambahan menggunakan *game framework* Cocos2d-x.

4.1 Lingkungan Pengembangan

Dalam membangun aplikasi ini digunakan beberapa perangkat pendukung baik perangkat keras maupun perangkat lunak. Lingkungan pembangunan dijelaskan sebagai berikut.

4.1.1 Lingkungan Pembangunan Perangkat Keras

Perangkat keras yang digunakan dalam pembuatan aplikasi ini adalah sebuah *desktop PC* dengan spesifikasi sebagai berikut.

- Prosesor Intel(R) Core i3 CPU @ 3,5GHz
- Memori (RAM) 4,00 GB
- Kartu Grafis NVIDIA GT 705 1 GB

4.1.2 Lingkungan Pembangunan Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan untuk membuat aplikasi ini sebaga berikut.

- Visual Studio Professional 2012
- Windows 8.1 Pro 64 bit Update 1 sebagai sistem operasi
- Cocos2d-x Version 3.3

- Tiled Map Editor Version 0.11.0
- TexturePacker Version 3.1.2
- PlistPad Version 0.1.0
- Graphics Gale Version 1.3.2
- Adobe Photoshop CS5

4.2 Implementasi Antarmuka

Pada subbab ini akan dibahas implementasi antarmuka berdasarkan rancangan antarmuka yang telah dibahas pada bab 3.

4.2.1 Implementasi Antarmuka Layar *Main Menu*

Layar *main menu* adalah tampilan yang pertama kali dilihat pemain saat membuka aplikasi permainan. Pada layar *main menu*, pemain dapat memilih tingkat kesulitan komputer yang akan dilawan dengan memilih *list menu* tingkat kesulitan. Berdasarkan rancangan yang telah dibuat di subbab 3.2.4.1, implementasi dari tampilan layar *main menu* ditunjukkan pada Gambar 4.1 dan Gambar 4.2.

Gambar 4.1 Tampilan Implementasi Antarmuka Layar *Main Menu*

Gambar 4.2 Tampilan Implementasi Antarmuka *Main Menu* Saat Memilih Tingkat Kesulitan

Gambar 4.1 adalah tampilan layar *main menu* saat pertama kali dibuka, dan Gambar 4.2 adalah saat pemain menekan tombol *play*.

4.2.2 Implementasi Antarmuka Layar *Battle*

Layar *battle* adalah layar yang dilihat pemain setelah emmilih tingkat kesulitan di layar *main menu*. Pada layar ini, pemain akan bertarung melawan pemain lain berdasarkan aturan permainan yang telah dirancang.

Dalam layar inilah pemain dapat membeli unit, menggerakkan unit, menyerang unit lawan, merebut bangunan, dan aksi lain dalam peraturan permainan. Tampilan implementasi dari layar *battle* ditunjukkan pada Gambar 4.3.

Gambar 4.3 Tampilan Implementasi Antarmuka Layar *Battle*

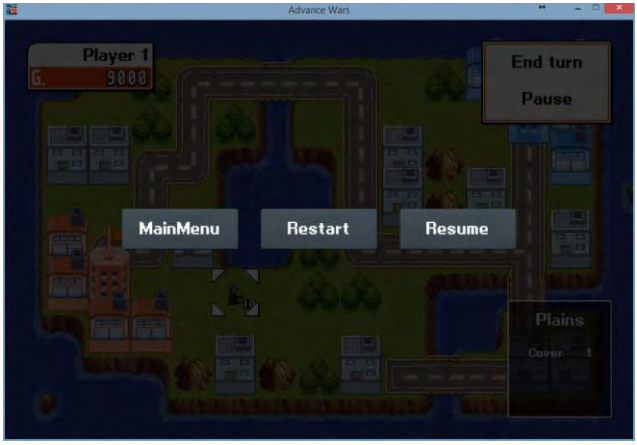
4.2.3 Implementasi Antarmuka Layar *Change Turn*

Layar *change turn* adalah tampilan yang muncul saat seorang pemain mengakhiri gilirannya dan berganti ke pemain lawannya, yaitu dengan memilih menu aksi "*end turn"*. Tampilan dari layar *change turn* ditunjukkan pada Gambar 4.4.

Gambar 4.4 Tampilan Implementasi Layar *Change Turn*

4.2.4 Implementasi Antarmuka Layar Pause

Layar *pause* adalah layar yang muncul jika pemain memilih aksi "*pause"* pada layar *battle*. Pada layar ini pemain dapat memilih untuk melanjutkan kembali permainan, mengulang permainan dengan tingkat kesulitan yang sama, atau keluar ke layar *main menu* dengan menekan tombol-tombol yag telah disediakan. Tampilan dari layar *pause* ditunjukkan pada Gambar 4.5.



Gambar 4.5 Tampilan Implementasi Antarmuka Layar *Pause*

4.2.5 Implementasi Antarmuka Window Buy Unit

Window buy unit adalah window atau jendela yang muncul ketika pemain memilih aksi "*buy unit*" pada layar *battle*. Pada window ini, pemain dapat melakukan pembelian *unit* yang akan digunakan dalam permainan. Tampilan dari *window buy unit* ditunjukkan pada Gambar 4.6.

Gambar 4.6 Tampilan Implementasi *Window Buy Unit*

4.2.6 Implementasi Antarmuka Window Game Over

Window game over adalah window yang muncul ketika salah satu pemain berhasil mengalahkan lawannya.

Gambar 4.7 Tampilan Implementasi Layar *Game Over*

Window ini berisi informasi-informasi jumlah sisa unit dan total *hitPoints* kedua pemain. Pemain dapat memilih untuk mengulang permainan dengan tingkat kesulitan yang sama atau kembali ke layar *main* menu dengan menekan tombol yang telah disediakan. Tampilan dari *window game over* ditunjukkan pada Gambar 4.7.

4.3 Implementasi Objek Permainan

Implementasi dari objek permainan yang telah dirancang pada subbab 3.2.1 yang berupa kode sumber asli terdapat pada lampiran. Tampilan untuk implementasi objek permainan ditunjukkan pada Gambar 4.8, Gambar 4.9, dan Gambar 4.10. Gambar 4.8 adalah tampilan untuk objek *terrain*. Gambar 4.9 adalah tampilan untuk objek unit, dan Gambar 4.10 adalah tampilan untuk objek *building*. Implementasi kode sumber untuk obejek permainan terdapat pada bagian lampiran. Implementasi kode sumber untuk objek terrain terdapat di Kode Sumber 7.1, implementasi kode sumber untuk objek unit di Kode Sumber 7.2, dan implementasi kode sumber untuk objek building di Kode Sumber 7.3.

Gambar 4.8 Tampilan Implementasi Objek *Terrain*

Gambar 4.9 Tampilan Implementasi Objek Unit

Gambar 4.10 Tampilan Implementasi Objek *Building*

4.4 Implementasi Aturan Main Permainan

Pada subbab ini akan dibahas implementasi mekanisme-mekanisme aturan main permainan yang telah dirancang pada subbab 3.2.2.

4.4.1 Implementasi Tahap Persiapan

Implementasi untuk tahap persiapan yang telah dirancang pada subbab 3.2.2.1 adalah fungsi *initializeBattle()*. Fungsi *initializeBattle()* akan menyiapkan semua kebuthan permainan seperti menyiapkan pemain, peta permainan, bangunan, unit dan

lain-lain. Implementasi untuk fungsi tahap persiapan ditunjukkan pada Kode Sumber 4.1.

```
void SceneBattle::initializeBattle() {
	playerTurn = 1;
	selectedUnit = nullptr;
	gameTurn = 0;
	p1UnitsKilled = 0;
	p2UnitsKilled = 0;
	p1UnitsBought = 0;
	p2UnitsBought = 0;

	this->initializeMap();
	this->initializeUnits();
	this->initializeCursor();
	this->initializeContextMenu();
	this->initializePlayers();
	this->displayHUD();
	this->initializeWindow();
	this->showEndTurnTransition();

	label = Label::createWithTTF(std::to_string
(unitsAttackThisTurn[1]).append(" - ").append
(std::to_string(unitsAttackThisTurn[2])),
"Fonts/aansa.ttf", 34, Size(160, 48),
TextHAlignment::CENTER);
	label->setColor(Color3B::WHITE);
	label->enableOutline(Color4B::BLACK,1);
	label->setPosition
(visibleSize.width/2,visibleSize.height - 100);
	this->addChild(label,101);
	this->scheduleUpdate();
	phase = PHASE_IDLE;
	unitsAttackThisTurn[0] = 0;
	unitsAttackThisTurn[1] = 0;
	unitsAttackThisTurn[2] = 0;
	isBattleReady = true;
}
```

Kode Sumber 4.1 Implementasi Tahap Persiapan

4.4.2 Implementasi Fase Siaga (*Standby Phase*)

Implementasi untuk fase siaga yang telah dirancang pada subbab 3.2.2.2.1 adalah fungsi *beginTurn()*.Fungsi *beginTurn()* adalah fungsi yang dipanggil ketika seorang pemain mendapatkan giliran. Implementasi untuk fase siaga ditunjukkan pada Kode Sumber 4.2. Tampilan ketika fase siaga ditunjukkan pada Gambar 4.11.

```
void SceneBattle::beginTurn(Player player) {
	gameTurn += 1;
	if(gameTurn > 100) {
	auto nextScene = SceneBattle::createScene();
	Director::sharedDirector()->replaceScene
(TransitionFade::create(1, nextScene));
	}

	playerScores[playerTurn] = 0;
	unitsMovedThisTurn[playerTurn] = 0;
	unitsCaptureThisTurn[playerTurn] = 0;
		hudGoldInfo->setVisible(true);
		activateUnits(player->Units);

		if(gameTurn <= 20)
		generateMoney(playerRed);
		showIdleMenu();

		if(player->getPlayerType() == 1){
			player->feedInformation(this);
		}
		for(int i = 0; i < player->
Units.size();i++){
		auto building = getBuildingInTile(
getTileDataAt(player->Units.at(i)-
>getGridPosition()));

		if(building!=nullptr) {
		if(building->getPlayer() == player){
```

```
                player->Units.at(i)-
>showSupplyAnimation();
                player->Units.at(i)->refillAmmoMax();
                player->Units.at(i)->refillFuelMax();
                player->Units.at(i)-
>replenishHitPoints(2);
                        }
                }
        }
}
```

Kode Sumber 4.2 Implementasi Fase Siaga

Gambar 4.11 Tampilan Implementasi Fase Siaga

4.4.3 Implementasi Membeli Unit

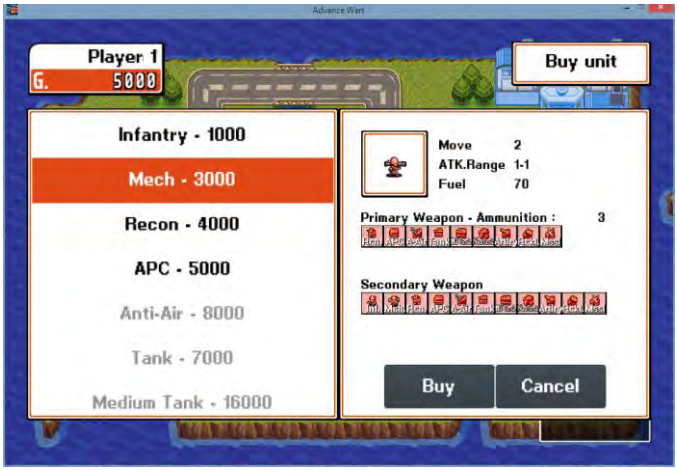
Implementasi untuk aksi membeli unit yang telah dirancang pada subbab 3.2.2.2.2 adalah fungsi *showBuildingMenu()* dan fungsi *checkIfBuyUnit()*. Fungsi *showBuildingMenu()* akan memeriksa apakah *factory* yang dipilih pemain kosong, untuk kemudian menampilkan *window buy unit*. Fungsi *checkIfBuyUnit()* dipanggil ketika pemain menekan tombol

"*buy*" pada *window buy unit*. Fungsi ini akan memeriksa apakah pemain memiliki uang yang sesuai dengan harga unit yang dipilihnya. Implementasi untuk aksi membeli unit ditunjukkan pada Kode Sumber 4.3. Tampilan ketika membeli unit ditunjukkan pada Gambar 4.12.

```
// FUNGSI showBuildingMenu()
void SceneBattle::showBuildingMenu(Building *
building) {
	hideContextMenu();
	Menu * contextMenu = Menu::create();
	contextMenu = menu_idleContextMenu;
	if(building != NULL) {
	if(building->getPlayer() == playerTurn) {
		if(building->getBuildingID() == 8) {
		contextMenu = menu_factoryContextMenu;
		}
	}
}
	contextMenu->setVisible(true); }

// FUNGSI checkIfBuyUnit()
void SceneBattle::checkIfBuyUnit() {
	if(playerTurn == 1) {
	if((playerRed->getGold() >= units.at(
windowBuyUnit->getSelectedUnit())->getCost()) &&
phase == PHASE_BUYUNIT) {
	windowBuyUnit->setIsBuy(false);

	int i = windowBuyUnit->getSelectedUnit();

	if(otherUnitInTile(getTileDataAt(tileCoordForP
osition( cursor->getPosition()))) == nullptr)
		buyUnit(i,cursor->getPosition());
		} }
	else if(playerTurn == 2) {
	if( (playerBlue->getGold() >= units.at(
windowBuyUnit->getSelectedUnit())->getCost()) &&
phase == PHASE_BUYUNIT) {
```

```
windowBuyUnit->setIsBuy(false);
int i = windowBuyUnit->getSelectedUnit();

        if(otherUnitIn Tile(get TileDataAt( tile CoordForP
osition( cursor->getPosition()))) == nullptr)
        buyUnit(i,cursor->getPosition());
                }
        }
}
```

Kode Sumber 4.3 Implementasi Membeli Unit



Gambar 4.12 Tampilan Implementasi Membeli Unit

4.4.4 Implementasi Memilih Unit

Implementasi untuk aksi memilih unit yang telah dirancang pada subbab 3.2.2.2.3 adalah fungsi *markPossilbeAction()*. Fungsi ini akan menampilkan *movement tile*, yaitu set *tile* yang dapat dilewati unit pemain saat dipilih.

Implementasi berupa *pseudocode* untuk aksi memilih unit ditunjukkan pada Kode Sumber 4.4. Untuk implementasi memilih unit yang berupa kode sumber aslinya terdapat di bagian lampiran,

di Kode Sumber 7.4. Tampilan ketika pemain memilih unit ditunjukkan pada Gambar 4.13.

```
void Function markPossibleAction() {

    dapatkan Tile Tempat Unit Berdiri

    masukkan tile tersebut ke movement tiles

    For each tile in tile list
    do
        dapatkan empat tile tetangga

        For each tile in tile tetangga
            pilih satu tile tetangga

            if sudah ada di closed list
                continue

            if ada unit lawan di tile tersebut
                continue

            if unit ini tidak bisa bergerak di tile
tersebut
                continue

            if unit ini bisa bergerak di tile
tersebut
                masukkan tile ke movement list

            if G Score tile tersebut > nilai attribut
movement point unit ini
                continue

            tambahakan tile tersebut ke open list
            tambahkan tile tersebut ke closed list

        i++
    while i < ukuran open list
```

```
    kosongkan closed list

    kosongkan open list
}
```

Kode Sumber 4.4 Implementasi Memilih Unit

Gambar 4.13 Tampilan Implementasi Memilih Unit

4.4.5 Implementasi Menggerakkan Unit

Implementasi untuk aksi menggerakkan unit yang telah dirancang pada subbab 3.2.2.2.4 adalah fungsi *doMarkedMovement()*. Fungsi ini akan menggerakkan unit ke *tile* yang telah dipilih sebagai tujuan gerakan dengan jarak terdekat dengan menggunakan algoritma pencarian jejak A* (*A* pathfinding algorithm*). Implementasi berupa *pseudocode* dari aksi menggerakkan unit ditunjukkan pada Kode Sumber 4.5. Sedangkan untuk implementasi berupa kode sumber aslinya

terdapat di bagian lampiran di Kode Sumbr 7.5. Tampilan ketika unit bergerak ditunjukkan pada Gambar 4.14.

```
void Function doMarkedMovement()
{
    dapatkan tile tempat unit ini berdiri
    do {
        ambil tile paling atas di open list

        if tile tersebut adalah tujuan
            gerakkan unit ini ke tile tersebut

        dapatkan 4 tile tetangga dari tile tersebut

        for each ntile in tile tetangga

            if ntile ada di closed list
                continue
            if  ada unit lawan di ntile
                continue
            if  tidak bisa bergerak di ntile
                continue

            dapatkan cost untuk bergerak ke ntile

            if ntile tidak ada di open list
                set tile sekarang sebagai parent
dari ntile
                set G score ntile = G score tile
sekarang + cost
                hitung H score untuk ntile
                masukkan ntile ke open list dan
urutkan
            else
                if tile G score sekarang + cost < G
score ntile
                    G score ntile = G score ntile +
cost
```

```
                    kosongkan open list
                    masukkan ntile ke open list dan
urutkan

    } while ukuran open list > 0
}
```

Kode Sumber 4.5 Implementasi Menggerakkan Unit

Gambar 4.14 Tampilan Implementasi Menggerakkan Unit

4.4.6 Implementasi Menyerang Unit Lawan

Implementasi aksi menyerang unit lawan yang telah dirancang pada subbab 3.2.2.2.5 adalah fungsi *attackTarget()* dan *applyDamage()*. Fungsi *attackTarget()* dipanggil ketika pemain memilih sebuah unit lawan setelah memilih menu "*attack*".

Gambar 4.15 Tampilan Implementasi Menyerang Unit Lawan

Fungsi ini akan memanggil fungsi *applyDamage()* pada unit lawan yang akan mengaplikasikan *damage* dari serangan unit pemain. Implementasi dari aksi menyerang unit lawan ditunjukkan pada Kode Sumber 4.6. Gambar 4.15 adalah tampilan ketika unit pemain menyerang unit lawan.

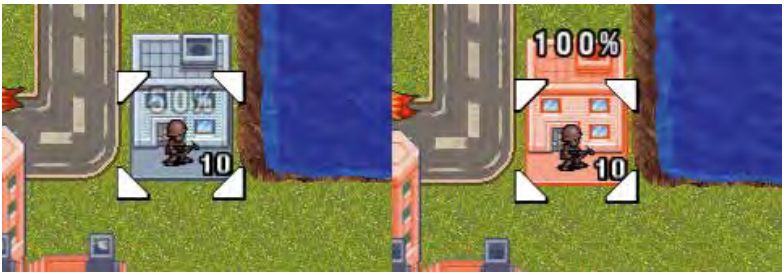
```
// FUNGSI attackTarget()
void Unit::attackTarget(Unit * target){
	int targetHP = target->getHitPoints();
	int attackDamage =
calculateAttackDamage(target);
	int counterDamage = 0;

	target->applyDamage(attackDamage,this);
	if(target->getCanCounter() == true && target->
isEnemyInAttackRange(target->getGridPosition(), this)
== true){
		target->counterTarget(this);
		unmarkPossibleAttack2();
	}
	unmarkPossibleAttack();

	if(isAttackUsingPrimary==true)
		ammunition -= 1;
	game->endAttack();
}

// FUNGSI applyDamage()
void Unit::applyDamage(int damage, Unit * attacker){
	if(damage <= 0) return;

	if(damage >= 10)
		int aaaaa= 0;

	this->hitPoints -= damage;

	this->labelHP->
setString(std::to_string(hitPoints));
	if(hitPoints <= 0){
```

```
                game->removeUnit(this,player);
                attacker->lockTarget(nullptr);
        }
}
```

Kode Sumber 4.6 Implementasi Menyerang Unit Lawan

4.4.7 Implementasi Mengambilalih Bangunan

Implementasi aksi mengambilalih bangunan yang telah dirancang pada subbab 3.2.2.2.6 adalah funsi *capture()*. Fungsi ini dipanggil ketika pemain memilih menu aksi "*capture*". Fungsi ini akan memeriksa apakah unit berdiri di sebuah bangunan dan dapat melakukan *capture*. Implementasi dari aksi mengambilalih bangunan ditunjukkan pada Kode Sumber 4.7. Tampilan ketika unit mengambilalih bangunan ditunjukkan pada Gambar 4.16.



Gambar 4.16 Tampilan Implementasi Mengambilalih Bangunan

```
void Unit::capture() {
        Building * building = game-
>getBuildingInTile(game->getTileDataAt(this-
>gridPosition));

        if(building->getPlayer() != this->player){
                if(building->getCanBeCaptured() ==
true) {
```

```
if(building->getCapturePoint() - this->hitPoints < 0)
	building->setCapturePoint(0);
else
	building->setCapturePoint(building->
getCapturePoint() - this->hitPoints);

	int captureDamage = (building->
getMaxCapturePoint()-building->getCapturePoint());
if(captureDamage < 0) captureDamage = 0;
	int capturePercentage = captureDamage * 100 /
building->getMaxCapturePoint();
	this->capturingBuilding = building;
	showCaptureAnimation(building,capturePercentag
e);
if(building->getCapturePoint() <= 0){
	building->setCapturePoint(0);
	if(building->getPlayer() == 1){
	game->p1Buildings.eraseObject(building);
	game->p2Buildings.pushBack(building);
} else if(building->getPlayer() == 2) {
	game->p2Buildings.eraseObject(building);
	game->p1Buildings.pushBack(building);
} else if(building->getPlayer() == 0) {
	if(this->player == 1)
	game->p1Buildings.pushBack(building);
	else if(this->player == 2)
	game->p2Buildings.pushBack(building);
}
	building->changePlayer(this->player);
			}
		}
	}
}
```

Kode Sumber 4.7 Implementasi Mengambilalih Bangunan

4.5 Implementasi Kecerdasan Buatan Adaptif

Pada subbab ini akan dibahas implementasi dari mekanisme-mekanisme kecerdasan buatan adaptif (*AI_Adaptive*) yang telah dirancang di subbab 3.2.5.1.

4.5.1 Implementasi Pembelian Unit

Implementasi pembelian unit pada *AI_Adaptive* yang telah dirancang pada subbab 3.2.5.1.1 adalah fungsi *buyBalancedUnitsAsPlayer*(). Fungsi ini akan berusaha membeli unit secara seimbang mungkin dengan pemain lawan. Implementasi dari mekanisme pembelian unit pada *AI_Adaptive* ditnjukkan pada Kode Sumber 4.8.

```
void AI_Adaptive::buyBalancedUnitAsPlayer() {
	this->unitsToBuy.clear();
	int enemyUnitsWorth = 0;
	int ownUnitsWorth = 0;
	for(int i = 0; i < enemyUnits.size();i++){
		enemyUnitsWorth += enemyUnits.at(i)->
getCost();
	}
	for(int i = 0; i < ownUnits.size();i++){
		ownUnitsWorth += ownUnits.at(i)->
getCost();
	}

	if(enemyUnits.empty()){
		game->endTurn();
		return;
	}
	else if(enemyUnitsWorth > ownUnitsWorth){
		for(int i = 0 ; i <
enemyUnits.size();i++){
	int ammount =
getTotalEnemyUnit(enemyUnits.at(i)->getUnitId());
```

```
        int temp =  getTotalAllyUnit(enemyUnits.at(i)-
>getUnitId());
        int temp2 =
getTotalInUnitsToBuy(enemyUnits.at(i)->getUnitId());

        if(ammount > (temp+temp2)){
        for(int j = 0; j < ammount-(temp+temp2); j++)
        {
                insertInOrder(enemyUnits.at(i));
        } } }
std::vector<int> factoriesOrder =
generateUniqueIntegerList((int)factories.size());

        for(int i = 0; i < unitsToBuy.size(); i++ ){
                for(int j = 0; j <
factoriesOrder.size(); j++){
                if((game->otherUnitIn Tile(game->
get Tile DataAt(factories.at(factoriesOrder.at(j))->
getGridPosition())) == nullptr) && (game->
otherEnemyUnitIn Tile(game->
get Tile DataAt(factories.at(factoriesOrder.at(j))->
getGridPosition()),playerSide) == nullptr)){
        if(gold >= unitsToBuy.at(i)->getCost()){
        game->buyUnit(unitsToBuy.at(i)->
getUnitId(),game->positionFor TileCoord(
factories.at(factoriesOrder.at(j))->
getGridPosition()));
                                                break;
        } } } } }
        moveNextUnit = true;
}
```

Kode Sumber 4.8 Implementasi Pembelian Unit *AI_Adaptive*

4.5.2 Implementasi Pergerakan dan Aksi Unit

Implementasi mekanisme pergerakan dan aksi unit *AI_Adaptive* yang telah dirancang pada subbab 3.2.5.1.2 adalah fungsi *moveUnitAdaptive()*. Fungsi ini dipangil setelah

AI_Adaptive selesai melakukan pembelian unit. Implementasi dari mekanisme pergerakan dan aksi unit milik *AI_Adaptive* berupa *pseudocode* ditunjukkan pada Kode Sumber 4.9. Sementara implementasi yang berupa kode sumber aslinya terdapat di bagian lampiran di Kode Sumber 7.6.

```
Void Function moveUnitAdaptive(Unit unit)
{
    if unit dapat bergerak di giliran ini
        cari unit lawan di jarak serangan
    if unit dapat menyerang unit lawan
    && jumlah serangan lawan > jumlah serangan
sendiri
        serang unit lawan tersebut
    else if berdiri di atas bangunan bukan milik
sendiri
        if bisa melakukan capture
        && jumlah capture lawan > jumlah capture
sendiri lakukan capture terhadap bangunan tersebut
    else
        bergerak ke tile dengan skor paling seimbang

}
```

Kode Sumber 4.9 Implementasi Pergerakan dan Aksi Unit *AI_Adaptive*

4.5.3 Implementasi Fungsi Evaluasi

Implementasi fungsi evaluasi milik *AI_Adaptive* yang telah dirancang di subbab 3.2.5.1.3 adalah fungsi *calculateEvaluationFunction()* yang di dalamnya ia akan memanggil fungsi *calculateEnemyPartialScore()* dan *calculateAllyPartialScore()*. Fungsi ini dipanggil pada saat sebuah unit milik setiap pemain selesai melakukan sebuah aksi. Implementasi fungsi evaluasi ditunjukkan oleh Kode Sumber 4.10.

```
float AI_Adaptive::calculateEvaluationFunction(Unit *
currentUnit, TileData * targetTile) {
	Vector<Building*> buildings;

	float score_AI = 0;
	float score_enemy = 0;
	float score_ally = 0;

	if(currentUnit->getPlayer() == 1) {
		enemyUnits = game->p2Units;
		ownUnits = game->p1Units;
		buildings = game->p1Buildings;
	} else if(currentUnit->getPlayer() == 2) {
		enemyUnits = game->p1Units;
		ownUnits = game->p2Units;
		buildings = game->p2Buildings;
	}

	for(int i = 0; i < enemyUnits.size(); i++) {
	score_enemy +=
calculateEnemyPartialScore(currentUnit,enemyUnits.at(
i),targetTile);
	}

	for(int i = 0; i < ownUnits.size(); i++) {
	if(i != enemyUnits.getIndex(currentUnit)){
	score_ally +=
calculateAllyPartialScore(currentUnit,ownUnits.at(i),
targetTile);
		}
	}

for(int i = 0; i < buildings.size(); i++) {
	if(buildings.at(i)->getBuildingID() == 8) {
	score_allyCity +=
calculateAllyCityPartialScore(currentUnit,buildings.a
t(i),targetTile);
		}
	}
```

```
        score_AI = (score_enemy) + (score_ally);

        return score_AI;
}
```

Kode Sumber 4.10 Implementasi Fungsi Evaluasi *AI_Adaptive*

4.5.4 Implementasi Pengaturan Tingkat Kesulitan

Implementasi pengaturan tingkat kesulitan milik *AI_Adaptive* yang telah dirancang di subbab 3.2.5.1.4 adalah fungsi *calculateEvennessScore()* dan *getmostEvenTile()*. Fungsi *calculateEvennessScore()* akan menghitung selisih antara skor *AI_Adaptive* dan pemain lawan yang kemudian akan dicari *tile* mana yang sesuai dengan menggunakan fungsi getMostEven*Tile*(). Implementasi pengaturan tingkat kesulitan berupa *pseudocode* ditunjukkan pada Kode Sumber 4.11. Sementara implementasi yang berupa kode sumber aslinya terdapat di bagian lampiran di Kode Sumber 7.7.

```
float Function calculateEvennessScore(Unit unit,
TileData targetTile) {
    evenness score = skor fungsi evaluasi di tile
target - (jumlah total skor fungsi evaluasi lawan /
jumlah unit lawan yang bergerak)
    return evenness score
}
TileData getMostEvenTile(Unit unit) {
    output tile = tile tempat unit berada
    for each tile in movement tiles
        set skor tile dengan calculateEvennessScore()
    if tingkat kesulitan easy
        ambil tile dengan skor paling kecil untuk
output tile
    else if tingkat kesulitan fair
        ambil tile dengan skor paling mendekati nol
untuk output tile
    else if tingkat kesulitan hard
```

```
        ambil tile dengan skor paling besar untuk
output tile
    return output tile
}
```

Kode Sumber 4.11 Implementasi Pengaturan Tingkat Kesulitan *AI_Adaptive*

4.6 Implementasi Kecerdasan Buatan Lawan

Pada subbab ini akan dibahas implementasi dari mekanisme-mekanisme kecerdasan buatan lawan yang telah dirancang di subbab 3.2.5.1. Semua kecerdasan buatan ini akan dijadikan uji coba melawan *AI_Adaptive*.

4.6.1 Implementasi Pembelian Unit

Implementasi pembelian unit milik semua kecerdasan buatan lawan adalah fungsi *buyUnits()*. Fungsi ini akan membeli unit-unit yang akan digunakan dalam permainan sesuai prioritasnya. Implementasi mekanisme pembelian unit milik kecerdasan buatan lawan ditunjukkan pada Kode Sumber 4.12.

```
void AI_Rush::buyUnits() {
       int goldTemp = this->gold;

       for(int j = 0; j < factories.size(); j++) {
       Vec2 factoryPosition = factories.at(j)->
getGridPosition();
       TileData* factoryTileData = game->
getTileDataAt(factoryPosition);
       Unit* unitOnFactory = game->
otherUnitInTile(factoryTileData);
       Unit* enemyOnFactory =  game->
otherEnemyUnitInTile(factoryTileData,playerSide);

       if(unitOnFactory == nullptr && enemyOnFactory
== nullptr) {
```

```
        if(this->gold >= game->units.at(5)->getCost())
{
        buyUnit(5,factoryPosition);
}
else if(this->gold >= game->units.at(1)->getCost()) {
        buyUnit(1,factoryPosition);
        continue;
        }
else if(this->gold >= (game->units.at(0)->getCost()))
{
        buyUnit(0,factoryPosition);
        continue;
                                }
                        }
                }
        moveNextUnit = true;
}
```

Kode Sumber 4.12 Implementasi Pembelian Unit Kecerdasan Buatan Lawan

4.6.2 Implementasi Pergerakan dan Aksi Unit *AI_Rush*

Implementasi pergerakan dan aksi unit *AI_Rush* yang telah dirancang pada subbab 3.2.5.2.2 adalah fungsi *moveUnitToNearestEnemyUnit*(). Fungsi ini akan selalu mencari unit lawan terdekat dan akan menyerangnya jika memungkinkan. Implementasi dari pergerakan dan aksi unit *AI_Rush* berupa *pseudocode* ditunjukkan pada Kode Sumber 4.13. Sementara implementasi yang berupa kode sumber aslinya terdapat di bagian lampiran di Kode Sumber 7.8.

```
void function moveUnitToNearestEnemyUnit(Unit unit) {

if unit dapat bergerak di giliran ini
        cari unit lawan terdekat

if ada unit lawan di daerah radius serangan
            serang unit lawan tersebut
```

```
else if sedang berdiri di atas bangunan bukan milik
sendiri && bisa melakukan capture
            capture bangunan ini

else
            bergerak ke tile dengan jarak serangan
terjauh dan jarak terdekat dengan unit lawan
}
```

Kode Sumber 4.13 Implementasi Pergerakan dan Aksi Unit *AI_Rush*

4.6.3 Implementasi Pergerakan dan Aksi Unit *AI_UnitOffence*

Implementasi pergerakan dan aksi unit *AI_UnitOffence* yang telah dirancang di subbab 3.2.5.2.3 adalah fungsi *moveToRandomEnemyUnit()*. Fungsi ini akan mencari unit lwan secara *random* untuk kemdian menyerangnya sampai kalah, sebelum berpindah ke unit lawan lain. Implementasi dari mekanisme pergerakan dan aksi unit *AI_UnitOffence* berupa *pseudocode* ditunjukkan pada Kode Sumber 4.14. Smentara implementasi yang berupa kode sumber aslinya terdapat di bagian lampiran di Kode Sumber 7.9.

```
void Function moveToRandomEnemyUnit(Unit unit)
{
    if unit dapat bergerak di giliran ini
        cari unit random
        if sudah mengunci unit lawan
        && unit lawan tersebut ada di daerah
jangkauan serangan
            serang unit lawan tersebut
        else if unit berada di luar daerah jangkauan
serangan
            bergerak ke posisi terdekat dengan unit
lawan tersebut
```

```
        else if sedang berdiri di atas bangunan bukan
milik sendiri
        && bisa melakukan capture
            capture bangunan ini
        else
            bergerak ke tile dengan jarak serangan
terjauh dan jarak
            terdekat dengan unit lawan
}
```

Kode Sumber 4.14 Implementasi Pergerakan dan Aksi Unit *AI_UnitOffence*

4.6.4 Implementasi Pergerakan dan Aksi Unit *AI_Defensive*

Implementasi pergerakan dan aksi unit *AI_Defensive* yang telah dirancang di subbab 3.2.5.2.4 adalah fungsi *moveUnitDefensive()*. Fungsi ini akan menggerakkan unit ke unit kawan terdekat jika ada dan bergerak ke *tile* dengan nilai attribut cover tertinggi jika tidak ada. Unit hanya akan menyerang jika diprovokasi. Implementasi dari mekanisme pergerakan dan aksi unit *AI_Defensive* ditunjukkan pada Kode Sumber 4.15.

```
void AI_Defensive::moveUnitDefensive(Unit * unit) {

       if(game->selectedUnit == nullptr)
              game->selectUnit(unit);

       auto currentUnit = unit;

       if(currentUnit->getMovedThisTurn() == false) {
       if(currentUnit->getIsMoving() == false) {
                     Unit * nearestAlly = nullptr;
                     unit->unmarkPossibleAttack2();
                     Unit * enemy = unit->
checkEnemyInAttackRange();
                     unit->unmarkPossibleAttack2();
```

```
                    if(enemy!=nullptr && currentUnit-
>isPossibleToAttack(enemy)) {
                    currentUnit->
doMarkedMovement(game->getTileDataAt(currentUnit->
getGridPosition()));
       currentUnit->attackTarget(enemy);
       unit->unmarkPossibleAttack2();
       } else {
       if(ownUnits.size() > 1) {
       if(ownUnits.getIndex(currentUnit) != 0)
       nearestAlly = currentUnit->
getNearestAllyUnit(); }

       if(nearestAlly != nullptr){
       moveToNearestAllyUnit(currentUnit,nearestAlly)
;      } else {
       moveToNearestHighestCoverTile(currentUnit);
                           } } }  }
       else {
              if(unitToMoveIndex < ownUnits.size()-1)
              unitToMoveIndex++;
       } }
```

Kode Sumber 4.15 Implementasi Pergerakan dan Aksi *AI_Defensive*

4.6.5 Implementasi Pergerakan dan Aksi Unit *AI_Balance*

Implementasi pergerakan dan aksi unit *AI_Balance* yang telah dirancang di subbab 3.2.5.2.5 adalah *fungsi moveBalanced()* yang akan memanggil fungsi *moveToRandomUnit()* dan *moveUnitDefensive()* seperti yang digunakan pada *AI_UnitOffence* dan *AI_UnitOffence*(). Implementasi dari mekanisme pergerakan dan aksi *AI_UnitOffence* ditunjukkan pada Kode Sumber 4.16.

```
void AI_Balance::moveBalanced() {

if(currentUnit->getUnitId() == 5) {
```

```
        moveToRandomEnemyUnit(currentUnit);
} else {
        int e = RandomHelper::random_int(0,1);
        if(e== 0)
        moveToRandomEnemyUnit(currentUnit);
        else
        moveUnitDefensive(currentUnit);
        }
}
```

Kode Sumber 4.16 Implementasi Pergerakan dan Aksi Unit *AI_Balance*

[Halaman ini sengaja dikosongkan]

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada aplikasi yang dikembangkan. Pengujian yang dilakukan adalah pengujian terhadap kebutuhan fungsional secara keseluruhan. Pengujian ini mengacu pada kasus penggunaan pada bab tiga. Pengujian dilakukan dengan beberapa skenario. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian pada bagian akhir bab ini.

5.1 Lingkungan Pembangunan

Dalam membangun aplikasi ini digunakan beberapa perangkat pendukung baik perangkat keras maupun perangkat lunak. Perangkat keras yang digunakan dalam pembuatan aplikasi ini adalah sebuah *desktop PC* yang memiliki spesifikasi sebagai berikut.

- Prosesor Intel(R) Core i3 CPU @ 3,5GHz
- Memori (RAM) 4,00 GB
- Kartu Grafis NVIDIA GT 705 1 GB

5.2 Skenario Pengujian

Skenario pengujian yang dilakukan dibagi menjadi dua, yaitu pengujian melawan kecerdasan buatan lain dan pengujian melawan pemain manusia. Kedua skenario pengujian akan mempertandingkan *AI_Adaptive* dengan lawannya dalam permainan. Dalam pengujian, setiap pihak memulai permainan dengan jumlah uang yang sama.

Dari hasil permainan untuk setiap tingkat kesulitan melawan masing-masing kecerdasan buatan lawan dan pemain manusia akan dibuat sebuah *histogram*. Pada *histogram* terdapat sumbu vertikal yang merupakan selisih sisa *healthPoint* semua unit milik *AI_Adaptive* dengan lawannnya dalam 50 permainan melawan

kecerdasan buatan lawan, dan 11 permainan melawan pemain manusia. Nilai positif berarti *AI_Adative* unggul pada saat permainan itu, dan sebaliknya nilai negatif berarti kecerdasan buatan lawannya lebih unggul.

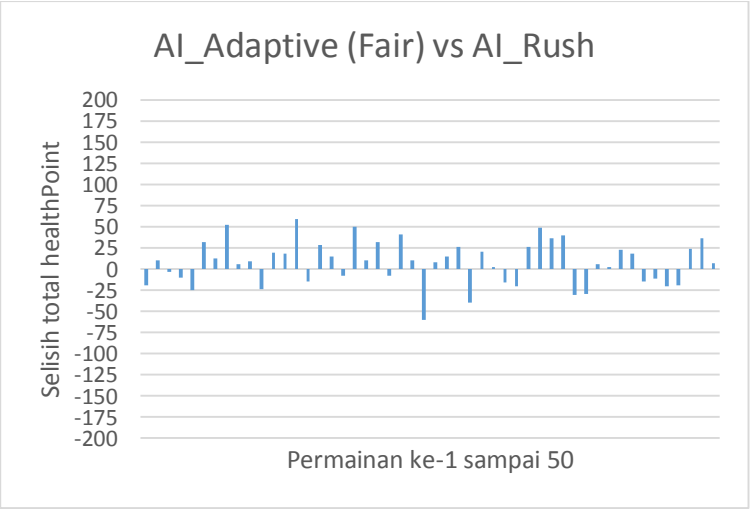
Selain *histogram*, dari hasil permainan untuk setiap tingkat kesulitan melawan masing-masing kecerdasan buatan lawan dan pemain manusia akan dibuat tabel yang berisi statistik hasil permainan. Statistik tersebut berisi nilai rata-rata sisa *healthPoint AI_Adaptive* setelah bermain untuk setiap permainan, nilai rata-rata sisa *healthPoint* ketika menang, dan ketika kalah. Ketiga nilai ini digunakan untuk melihat apakah *AI_Adaptive* sudah dapat bermain sesuai dengan tingkat kesulitan yang telah diatur kepadanya. Semakin besar nilainya berarti AI_Adaptive semakin mendominasi dalam semua permainan. Selain itu, terdapat nilai varian dan standar deviasi dalam tabel statistik hasil pengujian untuk melihat seberapa bervariasi dan stabil setiap set permainan. Semakin kecil berarti set permaian tersebut semakin stabil dan variasinya lebih sedikit.

5.2.1 Pengujian Melawan Kecerdasan Buatan Lain

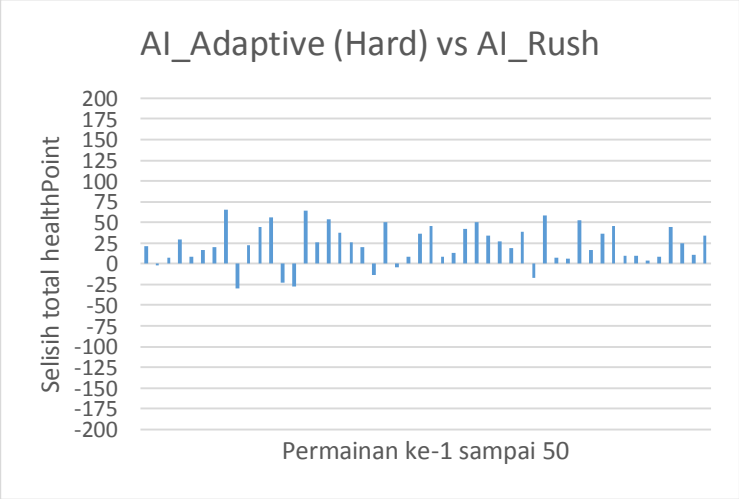
Pada pengujian melawan kecerdasan buatan lain, *AI_Adaptive* ditandingkan dalam permainan melawan kecerdasan buatan lawan yang telah dirancang dalam subbab 3.2.5.2. Pengujian dilakukan sebanyak 50 kali permainan untuk setiap tingkat kesulitan untuk setiap kecerdasan buatan lawan. Untuk mempercepat waktu pengujian, setiap permaianan dibatasi penghentian pasokan uang setelah 20 giliran.

5.2.1.1 Pengujian Melawan AI_Rush

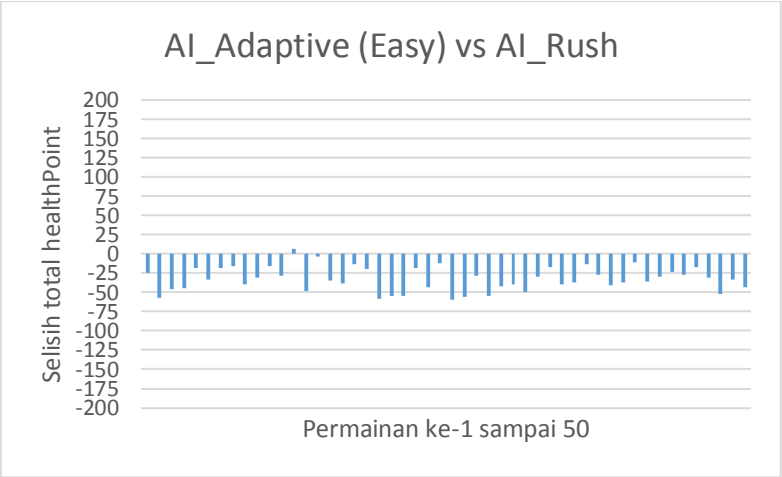
Dari pengujian antara *AI_Adaptive* dengan tingkat kesulitan fair, didapatkan hasil *histogram* seperti pada Gambar 5.1, *AI_Adaptive* dengan tingkat kesulitan *hard* pada Gambar 5.2, dan *AI_Adaptive* dengan kesulitan *easy* pada Gambar 5.3.



Gambar 5.1 Histogram Hasil Pengujian AI_Adaptive *Fair* Melawan AI_Rush



Gambar 5.2 Histogram Hasil Pengujian AI_Adaptive *Hard* Melawan AI_Rush



Gambar 5.3 Histogram Hasil Pengujian AI_Adaptive *Easy* Melawan AI_Rush

Tabel 5.1 Statistik Hasil Pengujian AI_Adaptive *Fair* Melawan AI_Rush

	Hard	**Fair**	**Easy**
Rata-Rata Menang	29.23	23.38	7.00
Rata-Rata Kalah	-16.57	-20.50	-33.59
Rata-Rata Gabungan	22.82	7.58	-32.78
Varian	546.80	671.88	245.77
Standar Deviasi	23.38	25.92	15.68

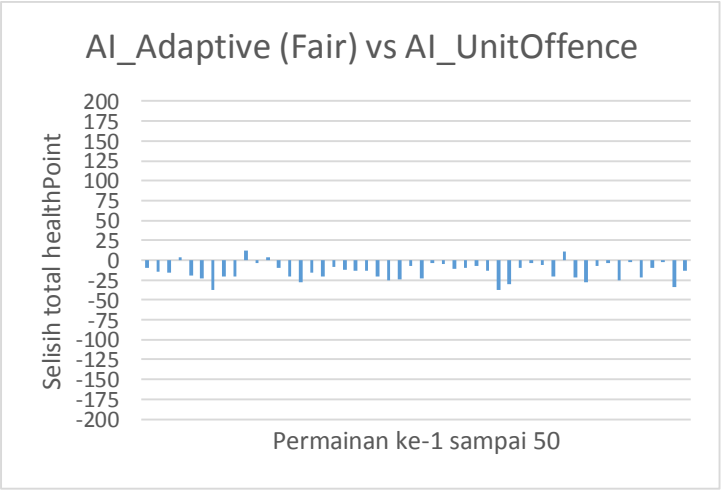
Dari hasil histogram pada Gambar 5.1 serta nilai varian dna standar deviasi pada Tabel 5.1, didapatkan bahwa *AI_Adaptive* dengan tingkat kesulitan *fair* memiliki variasi nilai beragam, namun sebagian besar nilai sudah mendekati nol. Hanya ada tiga permainan yang menghasilkan data di luar ambang 50/-50.

Sementara itu pada hasil pengujian milik *AI_Adaptive* dengan tingkat kesulitan *hard*, didapatkan hasil yang berbeda jauh. Dari Gambar 5.2 terlihat bahwa *AI_Adaptive* lebih sering unggul

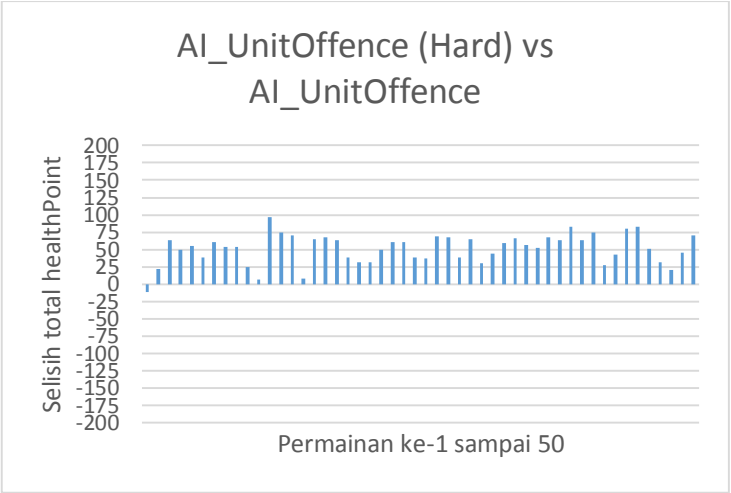
daripada *AI_Rush* pada tiap permaianannya. Sebaliknya, pada hasil pengujian milik *AI_Adaptive* dengan tingkat kesuitan *easy* pada Gambar.5.3, hampir semua permaianannya didominasi oleh *AI_Rush*, hanya terdapat satu permainan yang dimenangkan oleh *AI_Adaptive easy*.

5.2.1.2 Pengujian Melawan AI_UnitOffence

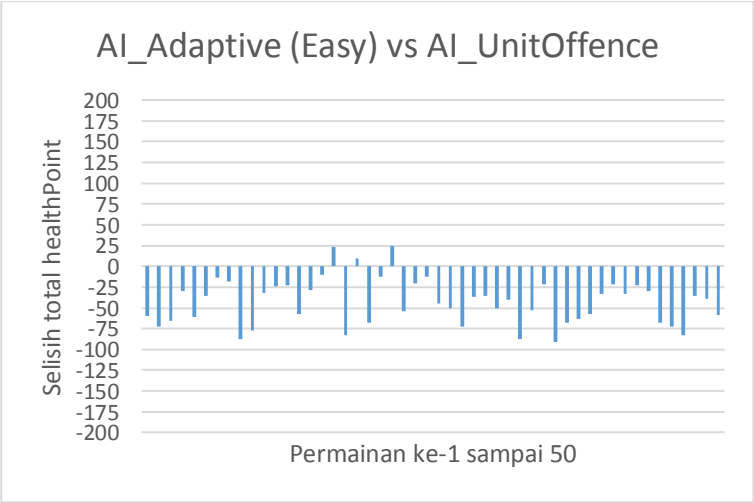
Dari pengujian melawan *AI_UnitOffence*, didapatkan hasil yang lebih stabil dibandingkan pada saat melawan *AI_Rush*. Pada pengujian *AI_Adaptive* dengan tingkat kesulitan fair, hampir semua permainan dimenangkan oleh *AI_UnitOffence*, namun nilai setiap pertandingan sangat mendekati nol. Hal ini menandakan *AI_Adaptive* dengan tingkat kesulitan fair dapat menyeimbangkan keadaan permainan melawan *AI_UnitOffence*. Sementara itu pada hasil pengujian *AI_Adaptive* dengan tingkat kesulitan *hard* dan *easy* tidak terlalu berbeda dengan hasil saat melawan *AI_Rush*.



Gambar 5.4 Histogram Hasil Pengujian AI_Adaptive *Fair* Melawan AI_UnitOffence



Gambar 5.5 Histogram Hasil Pengujian AI_Adaptive *Hard* melawan AI_UnitOffence



Gambar 5.6 Histogram Hasil Pengujian AI_Adaptive *Easy* Melawan AI_UnitOffence

Tabel 5.2 Statistik Hasil Pengujian AI_Adaptive Melawan AI_UnitOffence

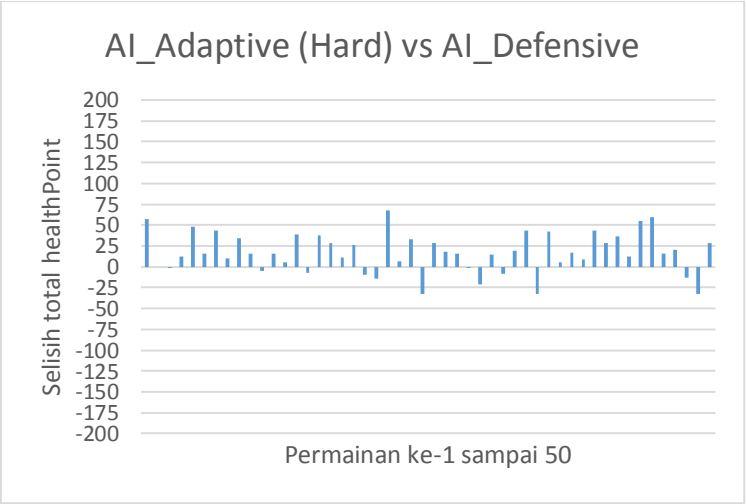
	Hard	***Fair***	***Easy***
Rata-Rata Menang	52.43	7.50	19.00
Rata-Rata Kalah	-11.00	-15.93	-47.26
Rata-Rata Gabungan	51.16	-14.06	-43.28
Varian	463.08	123.04	765.92
Standar Deviasi	21.52	11.09	27.68

5.2.1.3 Pengujian Melawan AI_Defensive

Pada pengujian antara *AI_Adaptive* dengan tingkat kesuliatn *fair* dan *easy* melawan *AI_Defensive*, semua unit miliknya tidak menyerang jika tidak diserang terlebih dahulu, sehingga pada saat pengujian, *AI_Adaptive* akan bertindak seimbang dengan ikut tidak menyerang lebih dulu, menjadikan permainan tidak pernah selesai. Sehingga penulis tidak mendapatkan data hasil pengujian. Meskipun setiap permainan melawan *AI_Defensive* dengan tingkat kesulitan tersebut tidak pernah selesai, hal ini bisa disimpulkan kalau *AI_Adaptive* telah bertindak benar dengan menjaga keadaan permainan yang seimbang. Untuk *AI_Adaptive* dengan tingkat kesulitan hard, karena ia memiliki inisiatif menyerang, penulis mendapatkan hasil pengujian, yang ditunjukkan oleh Gambar 5.7.

Tabel 5.3 Statistik Hasil Pengujian AI_Adaptive *Hard* Melawan AI_Defensive

	Hard
Rata-Rata Menang	27.62
Rata-Rata Kalah	-14.83
Rata-Rata Gabungan	16.88
Varian	576.6
Standar Deviasi	24.01



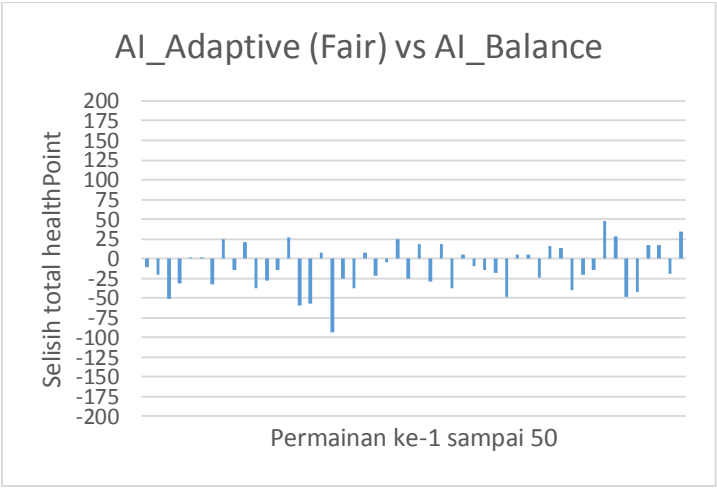
Gambar 5.7 Histogram Hasil Pengujian AI_Adaptive *Hard* Melawan AI_Defensive

Karena permainan melawan *AI_Defensive* yang berlangsung sangat lama, pemain sengaja menghentikan permainan pada gilirna ke 20 dan melihat selisih total healthPoint kedua pihak untuk mempersingkat waktu pengujian.

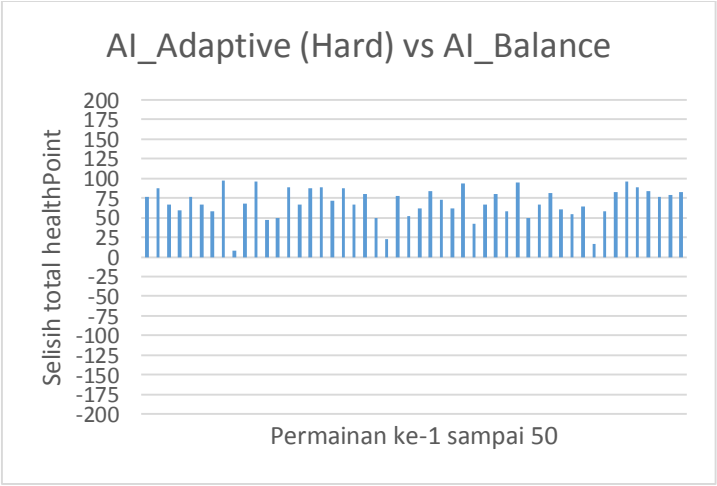
Dari data tersebut terlihat bahwa AI_Adaptive lebih unggul tipis daripada *AI_Defensive*.

5.2.1.4 Pengujian Melawan AI_Balance

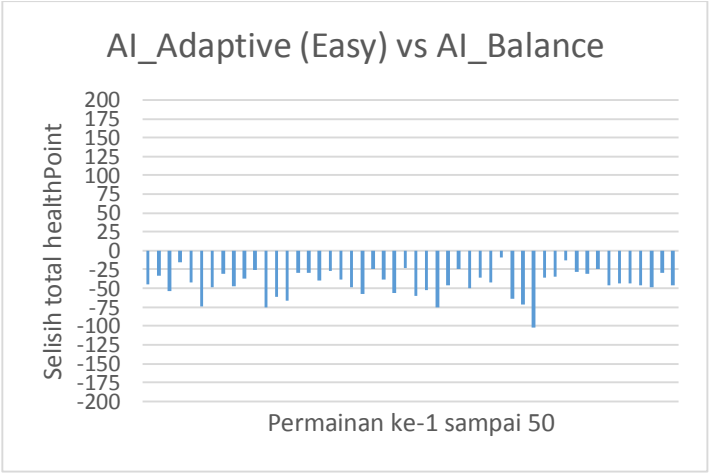
Dari pengujian *AI_Adaptive* melawan *AI_Balance*, didapatkan satatistik yang ditunjukkan pada Tabel 5.4 dan histogram yang ditunjukkan oleh Gambar 5.8, Gambar 5.9, dan Gambar 5.10.



Gambar 5.8 Histogram Hasil Pengujian AI_Adaptive *Fair* Melawan AI_Balance



Gambar 5.9 Histogram Hasil Pengujian AI_Adaptive *Hard* Melawan AI_Balance



Gambar 5.10 Histogram Hasil Pengujian AI_Adaptive *Easy* Melawan AI_Balance

Tabel 5.4 Statistik Hasil Pengujian AI_Adaptive Melawan AI_Balance

	Hard	***Fair***	***Easy***
Rata-Rata Menang	69.06	16.95	-
Rata-Rata Kalah	-	-31.13	-43.34
Rata-Rata Gabungan	69.06	-11.90	-43.34
Varian	394.87	828.91	325.70
Standar Deviasi	19.87	28.79	18.05

Dari Gambar 5.8, terlihat hasil yang tidak berbeda dari saat pengujian *AI_Adaptive fair* melawan *AI_Rush*, yaitu hasil yang berubah-ubah namun masih berusaha mendekati nol sekecil mungkin, terlihat dari satstik rata-rata menang, kalah, dan gabungan pada Tabel 5.5. Pada pengujian *AI_Adaptive* dengan tigkat kesulitan *hard*, terlihat bahwa semua permaianan berhasil dilawan oleh *AI_Adaptive*, sebaliknya untuk pengujian *AI_Adaptive* dengan tingkat kesulitan *easy* melawan *AI_Balance*,

semua permainan dimenangkan oleh *AI_Balance*. Sehingga tidak didapatkan nilai rata-rata kalah untuk tingkat kesulitan *hard* dan rata-rata menang untuk *easy*. Dari semua pengujian melawan kecerdasan buatan lain, dapat dikatakan bahwa *AI_Adaptive* dengan tingkat kesulitan *fair* berhasil bermain secara seimbang melawan semua kecerdasan buatan lain. Hal itu dilihat dari tidak adanya kecerdasan buatan yang dominan, baik *AI_Adaptive* maupun kecerdasan buatan yang lain. Sedangkan untuk *AI_Adaptive easy* dan *hard* terlihat bahwa *AI_Adaptive* dapat menyesuaikan permaian dengan tingkat kesulitan yang telah diatur.

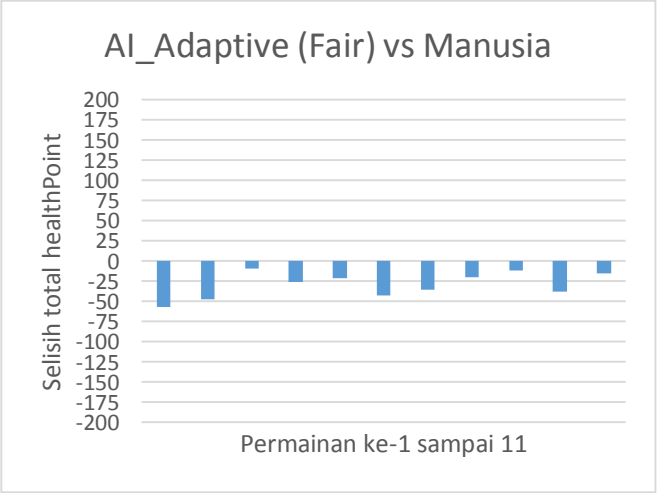
5.2.2 Pengujian Melawan Pemain Manusia

Pada pengujian melawan pemain manusia, *AI_Adaptive* ditandingkan dalam permainan melawan 11 pemain manusia. Daftar penguji dapat dilihat pada Tabel 5.5. Setiap pemain manusia akan bermain sekali melawan setiap tingkat kesulitan. Untuk mempercepat waktu pengujian, setiap permainan dibatasi penghentian pasokan uang setelah 20 giliran. Hasil dari pengujian ini ditunjukkan pada Gambar 5.11, Gambar 5.12, dan Gambar 5.13. Statistik untuk hasil pengujian ditunjukkan pada Tabel 5.6.

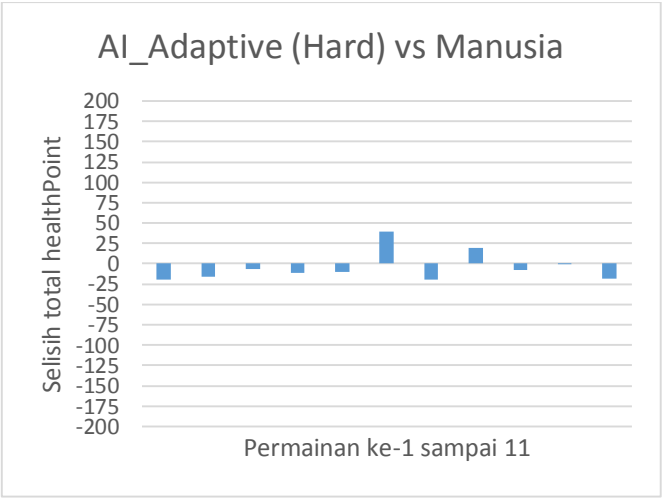
Tabel 5.5 Tabel Daftar Penguji

Nama Penguji	Pekerjaan
Luthfan Aufar	Mahasiswa
Fany Agriansyah	Mahasiswa
Ahmad Fauzi	Mahasiswa
Punggi Esthi Bawono	Mahasiswa
Julio Anthony Leonard	Mahasiswa
Herleeyandi Markoni	Mahasiswa
I Ketut Megi Trisnawan	Mahasiswa
Novandi Banitama	Mahasiswa
Hashfi Alfian Ciyuda	Mahasiswa
Mahardhika Maulana	Mahasiswa
Hifnie Bilfash	Mahasiswa

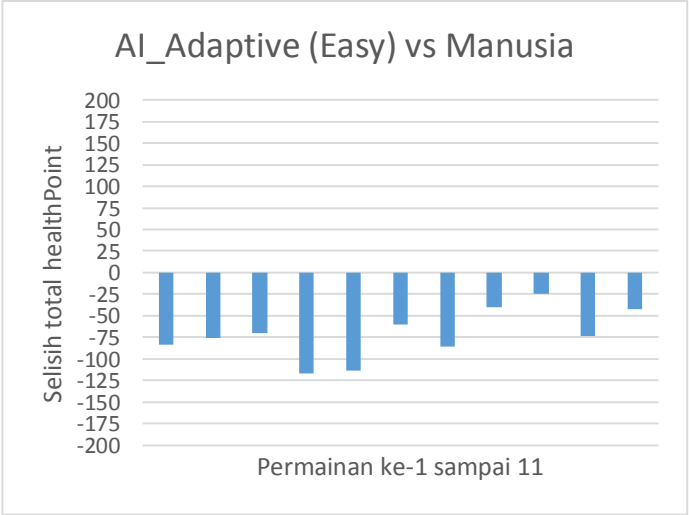
Dari hasil pengujian yang telah dilakukan terhadap pemain manusia. Terlihat bahwa *AI_Adaptive* hampir selalu kalah melawan pemain manusia dan memiliki selisih nilai *hitpoints* yang lebih besar jika dibandingkan dengan saat melawan sesama kecerdasan buatan. *AI_Adaptive* hanya menang dua permainan pada tingkat kesulitan *hard,* sehingga tidak didapatkan nilai rata-rata menang untuk tingkat kesulitan *fair* dan *easy* pada Tabel 5.7. Penulis mengamati dan menemukan bahwa pemain manusia memiliki kebiasaan untuk melakukan perencanaan taktik ke depan, misalnya dengan merencanakan sebuah jebakan. Hal ini belum dapat diantisipasi oleh fungsi evaluasi pada *AI_Adaptive* yang hanya mampu membaca keadaan permainan pada satu giliran setiap saat. Namun dari seluruh hasil pengujian melawan pemain manusia, didapatkan hasil bahwa *AI_Adaptive* masih dapat mempertahankan kemampuan adaptasi kemampuan lawannya.



Gambar 5.11 Histogram Hasil Pengujian AI_Adaptive *Fair* Melawan Pemain Manusia



Gambar 5.12 Histogram Hasil Pengujian AI_Adaptive Hard Melawan Pemain Manusia



Gambar 5.13 Histogram Hasil Pengujian AI_Adaptive Easy Melawan Pemain Manusia

Tabel 5.6 Statistik Hasil Pengujian *AI_Adaptive* Melawan Pemain Manusia

	Hard	Fair	Easy
Rata-Rata Menang	29	-	-
Rata-Rata Kalah	-12.33	-29.64	-71.64
Rata-Rata Gabungan	-4.82	-29.64	-71.64
Varian	334.16	246.65	-842.85
Standar Deviasi	18.28	15.71	29.03

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan mengenai uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran guna penyempurnaan kedepan.

6.1 Kesimpulan

Setelah dilakukan pengujian dan evaluasi terhadap hasil pengujian, maka dapat ditarik berbagai kesimpulan sebagai berikut:

1. Dari hasil pengujian, aplikasi permainan sudah dapat mengimplementasikan aturan main dari permainan *turn-based strategy* Advance Wars.
2. Implementasi fungsi evaluasi pada kecerdasan buatan adaptif sebagai pengevaluasi gerakan unit sudah dapat mengimbangi permainan lawannya, meskipun hasilnya tidak sebaik ketika melawan kecerdasan buatan lain. Hal itu dilihat dari hasil pengujian melawan pemain manusia yang selalu kalah, namun tetap memiliki nilai sisa *healthpoint* yang mendekati nol, di mana hal itu berarti pemain manusia tidak terlalu mendominasi dalam sebuah permainan.
3. Implementasi fungsi evaluasi dapat diatur tingkat kesulitannya dengan mengubah parameter pada saat proses kalkulasi perbandingan skor pemain dan kecerdasan buatan adaptif. Pada hasil pengujian melawan pemain manusia dan kecerdasan buatan lain, kecerdasan buatan adaptif dengan tingkat kesulitan *easy* lebih sering kalah dan memiliki nilai negatif lebih banyak dibanding tingkat kesulitan *fair* dan *hard*. Kecerdasan buatan adaptif dengan tingkat kesulitan hard memiliki nilai

positif lebih besar dan lebih sering menang dibandingkan tingkat kesulitan *fair* dan *easy*.

4. Performa kecerdasan buatan adaptif yang dibuat menggunakan fungsi evaluasi cukup baik dalam mengimbagi permainan kecerdasan buatan lain yang digunakan dalam permainan *turn-based strategy*. Pada semua pengujian melawan kecerdasan buatan lain, kecerdasan buatan adaptif selalu berusaha membuat nilai sisa *healthpoint* mendekati nol.

6.2 Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi, dan pengujian yang telah dilakukan.

1. Menambahkan mode *multiplayer* agar dapat dimainkan bersama pemain manusia lain.
2. Membuat versi perangkat *mobile* agar dapat dimainkan di manapun.
3. Menambahkan peta permainan (*map*) dinamis agar permainan lebih bervariasi setiap dimainkan.

DAFTAR PUSTAKA

[1] Ian Millington, "Artificial intelligence for Games". Morgan Kauffmann, 2006.

[2] Stuart J. Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach" (2nd ed.). Prentice Hall, 2003.

[3] Bill Loguidice and Matt Barton, "Vintage Games: An Insider Look at the History of Grand Theft Auto, Super Mario, and the Most Influential Games of All Time". Focal Press, 2009.

[4] K. Forbus and J. Laird, "AI and the entertainment industry," IEEE Intell. Syst., vol. 17, no. 4, pp. 15–16, Jul.-Aug. 2002.

[5] J. Schaeffer, "A gamut of games," Artif. Intell. Mag., vol. 22, no. 3, pp. 29–46, 2001.

[6] J.-B. Hoock, C.-S. Lee, A. Rimmel, F. Teytaud, M.-H. Wang, and O. Teytaud, "Intelligent agents for the game of go," IEEE Comput. Intell. Mag., vol. 5, no. 4, pp. 28–42, Nov. 2010.

[7] J. Hagelback and S. J. Johansson, "Measuring player experience on runtime dynamic difficulty scaling in an RTS game," in Proc. 5th Int. Conf. Comput. Intell. Games, 2009, pp. 46–52.

[8] Chin Hiong Tan, Kay Chen Tan, and Arthur Tay, "Dynamic Game Difficulty Scaling Using Adaptive Behavior-Based

AI", IEEE Transactions on Computational Intelligence and AI in Games Magazine, Vol.3, No.4, Dec. 2011.

[9] M. Bergsma and P. Spronck, "Adaptive intelligence for turn-based strategy games," in Proc. Belgian-Dutch Artif. Intell. Conf., 2008, pp. 17–24.

[10] B. D. Bryant and R. Miikkulainen, "Neuroevolution for adaptive teams," in Proc. IEEE Congr. Evol. Comput., 2003, vol. 3, pp. 2194–2201.

[11] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," IEEE Trans. Evol. Comput., vol. 9, no. 6, pp. 653–668, Dec. 2005.

[12] Steven Woodcock, "Game AI: The State of the Industry," [online]. Available: http://www.gamasutra.com/view/feature/131778/game_ai_the_state_of_the_industry.php. [Diakses 22 Mei 2015 18.37 WIB].

[13] Sander Bakkes, Pieter Spronck, Jaap van den Herik. Phase-dependent Evaluation in RTS Games. Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence, 2007, pp. 3-10.

[14] Remco Straatman, William van der Sterren, Arjen Beij. "Killzone's AI: dynamic procedural combat tactics". Game Developers Conference (GDC) 2005.

[15] Kittisak Potisartra and Vishnu Kotrajaras, “Towards an Evenly Match Opponent AI in Turn-based Strategy Games“, Chulalongkorn University, Thailand.

[16] Keith Burgun, “Game Design Theory: A New Philosophy for Understanding Games”. A K Peters/CRC Press, 2012.

[17] Fabian Fischer, “Criteria for Strategy Game Design,” 2014. [Online].Available: http://gamasutra.com/blogs/ FabianFischer/20141201/231243/Criteria_for_Strategy_Game_Design.php. [Diakses 22 Mei 2015 15.08 WIB].

[18] Gareth Davies, “Treatise on Combat to Pink Floyd,” Desember 2002. Available: http://www.rpgcodex.net/, [Diakses 22 Mei 2015 16.12 WIB].

[19] Mark Newheiser, “Playing Fair: A Look at Competition in Gaming”, 9 Maret 2009, Available: http://www.strangehorizons.com/2009/20090309/newheiser-a.shtml, [Diakses 15 Juni 2015 10.20 WIB].

[Halaman ini sengaja dikosongkan]

LAMPIRAN A KODE SUMBER

```
class TileData : public cocos2d::Node {
private:

	int terrainID;
	bool isSelectedForMovement;
	bool isSelectedForAttack;
	String tileType;
	int cover;
	int movementCost;
	Point position;
	TileData * parentTile;
	TileData * secondParentTile;
	int hScore;
	int fScore;
	int gScore;

public:
	TileData(void);
	~TileData(void);

};
```

Kode Sumber 7.1 Implementasi Objek *Terrain*

```
class Unit : public cocos2d::Node
{
private:
	int unitId;
	String name;
	int unitType;
	int cost;
	int movementPoints;
	int hitPoints;
	int maxHitPoints;
	int fuel;
	int maxFuel;
	int ammunition;
```

```
        int maxAmmunition;
        int minAttackRange;
        int maxAttackRange;
        bool canActAfterMoving;
        bool canCounter;
        bool canCapture;

        int player;
        AnimatedGameObjects *idleSprite;
        AnimatedGameObjects *moveSprite;
        AnimatedGameObjects *fireSprite;
        AnimatedGameObjects *inactiveSprite;
        AnimatedGameObjects *currentSprite;

        Label * labelHP;
        Point cameraPosition;
        Point worldPosition;
        Point gridPosition;

        int idleFrame;
        int moveFrame;

        Vector<TileData*> openSteps;
        Vector<TileData*> closedSteps;
        Vector<TileData*> movementPath;
        Vector<TileData*> movementTiles;

        bool isMoving;
        bool finishedMoving;
        bool movedThisTurn;
        bool attackedThisTurn;
        bool selectingMovement;
        bool selectingAttack;
        bool isAttackUsingPrimary;
        bool isEnableToCapture();
        Vec2 initialPosition;
        Vec2 positionBeforeMovement;
        TileData * tileDataBeforeMovement;
        int costAfterMovement;
        Unit * lockedTarget;
```

```
        Building * capturingBuilding;
public:
        Unit(void);
        ~Unit(void);

        std::vector<int> primaryDamage;
        std::vector<int> secondaryDamage;
        std::vector<int> movementCostOverride;

        void selectUnit();
        void unselectUnit();
        void unMarkPossibleMovement();
        void markPossibleAction(int action);
        void markMovementRange(int action);
        void unmarkMovementRange();
        void unmarkPossibleAttack();

        void insertOrderedinOpenSteps(TileData *
tile);
        int computeHScoreFromCoord(Vec2 fromCoord,
Vec2 toCoord);
        int costToMoveFromTile(TileData * fromTile,
TileData * toTile);
        void
constructPathAndStartAnimationFromStep(TileData *
tile);
        void popStepAndAnimate();
        void doMarkedMovement(TileData *
targetTileData);

        virtual bool init();

        bool isPossibleToAttack(Unit * target);
        bool canAttack();
        void attackUnit();
        void cancelAttackUnit();
        void attackTarget(Unit * target);
        void counterTarget(Unit * target);
        void applyDamage(int damage, Unit * attacker);
        int calculateAttackDamage(Unit * defender);
```

```
        int calculateCounterDamage(Unit * defender);
        void removeDamageLabel(Node * n);
        bool isEnemyInAttackRange(Vec2 standingPos,
Unit * unit);

        void setDamageAgainst(int unitType, int
primaryDamage, int secondaryDamage);
        void overrideMovementCostForTerrainType(int
terrainType,int movementCost);
        void overrideVisionForTerrainType(int
terrainType, int vision);
        void canTransport(int unitType, int slots)
        int getPrimaryDamageAgainst(int unitType);
        int getSecondaryDamageAgainst(int unitType);

        void createAnimatedSprite();
        void initializeLabelHP();
        void idle();
        void move();
        void inactive();
        void startTurn();
        void capture();

        void refillAmmo(int ammount);
        void refillAmmoMax();
        void refillFuel(int ammount);
        void refillFuelMax();
        void replenishHitPoints(int ammount);
        void replenishHitPointsMax();
        void showSupplyAnimation();
        void removeSupplyLabel(Node * n);

        void clearOpenClosedSteps();
        CREATE_FUNC(Unit);
};
```

Kode Sumber 7.2 Implementasi Objek Unit

```
class Building : public cocos2d::Node
{
protected:
	int player;
	int buildingID;
	String name;
	int cover;
	int movementCost;
	AnimatedGameObjects *currentSprite;
	AnimatedGameObjects *p1AnimatedSprite;
	AnimatedGameObjects *p2AnimatedSprite;
	Sprite *sprite;
	Sprite *neutralSprite;
	String spriteFileName;
	String p1spriteFileName;
	String p2spriteFileName;
	Sprite *p1Sprite;
	Sprite *p2Sprite;
	Vec2 gridPosition;
	Vec2 screenPosition;
	Vec2 worldPosition;
	bool isAnimated;
	bool canBeCaptured;
	int maxCapturePoint;
	int capturePoint;
public:

	Building(void);
	~Building(void);
	Building(String name, int owner, bool
isAnimated);
	virtual bool init();
	void changePlayer(int player);
	void resetCapturePoint();
	void generateMoney();
	void changeOwner(int player);
};
```

Kode Sumber 7.3 Implementasi Objek *Building*

```
void Unit::markPossibleAction(int action)
{

	initialPosition = this->gridPosition;
	TileData * startTileData = game-
>getTileDataAt(gridPosition);
	openSteps.pushBack(startTileData);
	closedSteps.pushBack(startTileData);

	movementTiles.pushBack(startTileData);

	int i = 0;
	if(fuel > 0)
	{
	do
	{
		TileData * currentTile =
openSteps.at(i);
		std::vector<Vec2> tiles = game-
>getTilesNextToTile(Vec2(currentTile-
>getPosition()));
		for(int j = 0; j < tiles.size(); j++) {
		TileData * neighborTile = game-
>getTileDataAt(tiles.at(j));

		bool valid = false;

	if(closedSteps.contains(neighborTile)) {
		continue; }


	if(game->
otherEnemyUnitInTile(neighborTile,player) != nullptr)
	{continue; }

	if(neighborTile->getMovementCost() >
movementPoints) {
	if(movementCostOverride.at(neighborTile-
>getTerrainID()) > 0 &&
```

```
movementCostOverride.at(neighborTile->getTerrainID())
<= movementPoints) {valid = true; }
	else {continue; }
}
	valid = true;
	if(valid == true)
	{
		neighborTile->setParentTile(nullptr);
		neighborTile->
setParentTile(currentTile);

	movementTiles.pushBack(neighborTile);

	if(fuel < movementPoints) {
		if(neighborTile->getGScore(this) >
fuel) {continue; }
		}
	else {
		if(neighborTile->getGScore(this) >
movementPoints) {continue; }
		}
	}
	openSteps.pushBack(neighborTile);
	closedSteps.pushBack(neighborTile);
				}
			}
			i++;
		} while (i < openSteps.size());
	}

		for(int a = 0; a <
movementTiles.size();a++){
		game-
>paintMovementTile(movementTiles.at(a));
}
	closedSteps.clear();
	openSteps.clear();
};
```

Kode Sumber 7.4 Implementasi Memilih Unit

```
void Unit::doMarkedMovement(TileData *
targetTileData) {
	costAfterMovement = 0;
	auto u = game->
otherUnitInTile(targetTileData);
	if( u != nullptr) {
		if(u != this)
			return;
	}

	if(isMoving)
		return;

	Vec2 startTile = game-
>tileCoordForPosition(worldPosition);
	tileDataBeforeMovement = game-
>getTileDataAt(startTile);
	insertOrderedinOpenSteps(tileDataBeforeMovemen
t);

	do{
		TileData * currentTile =
openSteps.at(0);
		Vec2 currentTileCoord = currentTile-
>getPosition();
		closedSteps.pushBack(currentTile);
		if(!openSteps.empty())
			openSteps.erase(0);
	if(currentTile-
>getPosition().equals(targetTileData-
>getPosition())){

constructPathAndStartAnimationFromStep(currentTile);
	openSteps.clear();
	closedSteps.clear();
	break;
	}
	std::vector<Vec2> tiles = game-
>getTilesNextToTile(currentTileCoord);
	for(int i = 0; i < tiles.size(); i++){
```

```
                        bool valid = false;
                        Vec2 tileCoord = tiles.at(i);
                        TileData * neighborTile = game-
>getTileDataAt(tileCoord);

        if(closedSteps.contains(neighborTile)) {
                continue; }
        if(game-
>otherEnemyUnitInTile(neighborTile,player)) {
                continue; }
        if(neighborTile->getMovementCost() >
movementPoints) {
        if(movementCostOverride.at(neighborTile-
>getTerrainID()) > 0 &&
movementCostOverride.at(neighborTile->getTerrainID())
<= movementPoints) {valid = true; }
        else {continue; }
}
        valid = true;
        if(valid == true) {
                int moveCost = this-
>costToMoveFromTile(currentTile,neighborTile);
                int index =
openSteps.getIndex(neighborTile);
                if(index == -1) {
                        neighborTile-
>setParentTile(nullptr);
                        neighborTile-
>setParentTile(currentTile);
                        neighborTile->gScore =
currentTile->gScore + moveCost;
                        neighborTile->setHScore(this-
>computeHScoreFromCoord(neighborTile-
>getPosition(),targetTileData->getPosition()));
                        this-
>insertOrderedinOpenSteps(neighborTile); } else {
        neighborTile = openSteps.at(index);

        if((currentTile->getGScore() + moveCost) <
neighborTile->getGScore()){
```

```
        neighborTile->setGScore(currentTile->gScore +
moveCost);

        if(!openSteps.empty())
        openSteps.erase(index);
        this->insertOrderedinOpenSteps(neighborTile);
                                        }
                                }
                        }

                }
        } while (openSteps.size() > 0);
}
```

Kode Sumber 7.5 Implementasi Menggerakkan Unit

```
void AI_Adaptive::moveUnitAdaptive(Unit * unit){

        if(game->selectedUnit == nullptr)
                game->selectUnit(unit);
        auto currentUnit = unit;
        if(currentUnit->getMovedThisTurn() == false) {
        if(currentUnit->getIsMoving() == false)

        currentUnit->unmarkPossibleAttack2();
        Unit * enemy = currentUnit-
>checkEnemyInAttackRange();
        auto * building = game-
>getBuildingInTile(game->getTileDataAt(currentUnit-
>getGridPosition()));
        currentUnit->unmarkPossibleAttack2();

        int enemySide = 0;
        if(this->playerSide == 1)
                enemySide = 2;
        else if(this->playerSide == 2)
                enemySide =1;
        if(enemy!=nullptr  && currentUnit-
>isPossibleToAttack(enemy) == true && game-
>unitsAttackThisTurn[enemySide] > game-
>unitsAttackThisTurn[playerSide]) {
```

```
        currentUnit->doMarkedMovement(game-
>getTileDataAt(currentUnit->getGridPosition()));
        currentUnit->attackTarget(enemy);
        currentUnit->unmarkPossibleAttack2(); }
        else if(building != nullptr && currentUnit-
>getCanCapture() == true && building->getPlayer() !=
this->playerSide && building->getCapturePoint() > 0
&& game->unitsCaptureThisTurn[playerSide] < game-
>unitsCaptureThisTurn[enemySide]) {
        currentUnit->capture();
        game->unitsCaptureThisTurn[playerSide] += 1;
currentUnit->setMovedThisTurn(true);
        currentUnit->setFuel(currentUnit->getFuel() -
currentUnit->getCostAfterMovement());
        currentUnit->inactive();
        currentUnit->setZOrder(currentUnit-
>getGridPosition().y + 1);
        }else{
        TileData * tileToMove =
getMostEvenTile(currentUnit);
        if(tileToMove != nullptr) {
                currentUnit->unMarkPossibleMovement();
                currentUnit-
>doMarkedMovement(tileToMove); }
        else{
                currentUnit->unMarkPossibleMovement();
                currentUnit->doMarkedMovement(game-
>getTileDataAt(currentUnit->getGridPosition()));
                }
        currentUnit->unmarkPossibleAttack2();
                        }
                }
        }else{
                if(unitToMoveIndex < ownUnits.size()-1)
                unitToMoveIndex++;
        }
}
```

Kode Sumber 7.6 Implementasi Pergerakan dan Aksi Unit *AI_Adaptive*

```
TileData * AI_Adaptive::getMostEvenTile(Unit *
currentUnit){
	TileData * evenTile = game-
>getTileDataAt(currentUnit->getGridPosition());
	currentUnit->unMarkPossibleMovement();
	currentUnit->unmarkMovementRange();
	currentUnit->markPossibleAction(3);
	Vector<TileData*>moveTiles;
	for(int i = 0; i < currentUnit-
>getMovementTiles().size();i++){
	if(!game->otherUnitInTile(currentUnit-
>getMovementTiles().at(i))) {
		moveTiles.pushBack(currentUnit-
>getMovementTiles().at(i));
		}
	}
	currentUnit->unMarkPossibleMovement();
	std::vector<float> scores;
	for(int i = 0; i < moveTiles.size(); i++){
	float x =
calculateEvennessScore(currentUnit,moveTiles.at(i));
		scores.push_back(x);
	}
	if(scores.empty())
		return game->getTileDataAt(currentUnit-
>getGridPosition());}

	int tileIndex = 0;
	int difficulty = 0;
	if(this->playerSide == 1)
		difficulty =
GameManager::selectedP1Level;
	else if(this->playerSide == 2)
		difficulty =
GameManager::selectedP2Level;

	evenTile = moveTiles.at(0);

	switch (difficulty)
	{
```

```
        case 0:
                //============== Easy
                for(int i = 0; i < scores.size(); i++)
                {
                        if( scores.at(i) <
scores.at(tileIndex))
                                tileIndex = i;
                }
                break;
        case 2:
                //============== Fair
                for(int i = 0; i < scores.size(); i++)
                {
                        if(abs( scores.at(i)) < abs(
scores.at(tileIndex)))
                                tileIndex = i;
                }
                break;
        case 4:
                //============== Hard
                for(int i = 0; i < scores.size(); i++)
                {
                        if( scores.at(i) >
scores.at(tileIndex))
                                tileIndex = i;
                }
                break;
        }
        evenTile = moveTiles.at(tileIndex);
        moveTiles.clear();
        return evenTile;
}
```

Kode Sumber 7.7 Implementasi Pengaturan Tingkat Kesulitan

```
void AI_Rush::moveUnitToNearestEnemyUnit(Unit * unit)
{
        if(game->selectedUnit == nullptr)
                game->selectUnit(unit);
```

```
        auto currentUnit = unit;
        if(currentUnit->getMovedThisTurn() == false)
        {
                if(currentUnit->getIsMoving() == false)
                {

                        auto target = currentUnit-
>getNearestEnemyUnit();
                        auto * building = game-
>getBuildingInTile(game->getTileDataAt(currentUnit-
>getGridPosition()));
                        if(target!=nullptr) {
                                currentUnit-
>unmarkPossibleAttack2();
                        if(currentUnit-
>isEnemyInAttackRange(currentUnit-
>getGridPosition(),target) == true && currentUnit-
>isPossibleToAttack(target)) {
                        currentUnit-
>unmarkPossibleAttack2();
                        currentUnit-
>doMarkedMovement(game->getTileDataAt(currentUnit-
>getGridPosition()));
                        currentUnit-
>attackTarget(target);
                        }
                        else if(building != nullptr &&
currentUnit->getCanCapture() == true && building-
>getPlayer() != this->playerSide && building-
>getCapturePoint() > 0) {
                        currentUnit->capture();
        game->unitsCaptureThisTurn[playerSide] += 1;
        game-
>calculateEvaluationFunction(currentUnit);
        currentUnit->setMovedThisTurn(true);
        currentUnit->setFuel(currentUnit->getFuel() -
currentUnit->getCostAfterMovement());
        currentUnit->inactive();
        currentUnit->setZOrder(currentUnit-
>getGridPosition().y + 1); }else{
```

```
        TileData * targetMovementTile = nullptr;
        int distanceIncrement = 0;
        do{
                targetMovementTile = currentUnit-
>getNearestEnemyUnitPosInAttackRange(target,distanceI
ncrement);
        distanceIncrement++;
        } while(targetMovementTile == nullptr);

        TileData * tileToMove = nullptr;
        if(targetMovementTile!=nullptr)
        tileToMove = currentUnit-
>getMaximumPosInMovementRangeTo(targetMovementTile);

if(tileToMove != nullptr)
        currentUnit->doMarkedMovement(tileToMove);
        else{
                currentUnit->unMarkPossibleMovement();
                currentUnit->doMarkedMovement(game-
>getTileDataAt(currentUnit->getGridPosition()));
                }
        currentUnit->unmarkPossibleAttack2();
                                }
                        }
                }
        }else{
                if(unitToMoveIndex < ownUnits.size()-1)
                unitToMoveIndex++;
        }
}
```

Kode Sumber 7.8 Implementasi Pergerakan dan Aksi Unit *AI_Rush*

```
void AI_UnitOffence::moveToRandomEnemyUnit(Unit *
unit) {
        if(game->selectedUnit == nullptr)
                game->selectUnit(unit);
```

```
        auto currentUnit = unit;

        if(currentUnit->getMovedThisTurn() == false) {
        if(currentUnit->getIsMoving() == false) {
        int randomModifier = 0;
        if(!enemyUnits.empty()){
        int maxRandom = enemyUnits.size() - 1;
        randomModifier =
RandomHelper::random_int(0,maxRandom); }
        Unit * target = nullptr;
        if(currentUnit->getLockedTarget() == nullptr {
                currentUnit-
>lockTarget(enemyUnits.at(randomModifier)); }

        target = currentUnit->getLockedTarget();
                auto * building = game-
>getBuildingInTile(game->getTileDataAt(currentUnit-
>getGridPosition()));

        if(target != nullptr) {
                currentUnit->unmarkPossibleAttack2();
        if(currentUnit-
>isEnemyInAttackRange(currentUnit-
>getGridPosition(),target) == true && currentUnit-
>isPossibleToAttack(target)) {
                currentUnit->unmarkPossibleAttack2();
                currentUnit->doMarkedMovement(game-
>getTileDataAt(currentUnit->getGridPosition()));
                currentUnit->attackTarget(target);
} else if(building != nullptr && currentUnit-
>getCanCapture() == true && building->getPlayer() !=
this->playerSide && building->getCapturePoint() > 0){
                currentUnit->capture();
        game->unitsCaptureThisTurn[playerSide] += 1;
        game-
>calculateEvaluationFunction(currentUnit);
        currentUnit->setMovedThisTurn(true);
        currentUnit->setFuel(currentUnit->getFuel() -
currentUnit->getCostAfterMovement());
        currentUnit->inactive();
```

```
        currentUnit->setZOrder(currentUnit-
>getGridPosition().y + 1);
        }else{
        TileData * targetMovementTile = nullptr;
                int distanceIncrement = 0;
                currentUnit->unMarkPossibleMovement();
                do{
                targetMovementTile = currentUnit-
>getNearestEnemyUnitPosInAttackRange(target,distanceI
ncrement);

        distanceIncrement++;}
while(targetMovementTile == nullptr &&
distanceIncrement <= 100);

        TileData * tileToMove = nullptr;
        if(targetMovementTile!=nullptr)
                tileToMove = currentUnit-
>getMaximumPosInMovementRangeTo(targetMovementTile);
        if(tileToMove != nullptr)
                currentUnit-
>doMarkedMovement(tileToMove);
        else{
                currentUnit->unMarkPossibleMovement();
                currentUnit->doMarkedMovement(game-
>getTileDataAt(currentUnit->getGridPosition()));
                }
        currentUnit->unmarkPossibleAttack2();
                                }
                        }
                }
        }
        else{
                if(unitToMoveIndex < ownUnits.size()-1)
                unitToMoveIndex++;
        }
}
```

Kode Sumber 7.9 Implementasi Pergerakan dan Aksi Unit AI_UnitOffence

BIODATA PENULIS

Penulis, Muhammad Arif Rohman Hakim lahir di Pekalongan, Jawa Tengah, pada tanggal 27 Maret 1993. Penulis adalah anak 1 dari 2 bersaudara dan dibesarkan di Kota Pekalongan

Penulis menempuh pendidikan formal di SD Negeri 11 Pekalongan (1999-2005), SMP Negeri 02 Pekalongan (2005-2008), dan SMA Negeri 01 Pekalongan (2008-2011). Pada tahun 2011, penulis menempuh pendidikan S1 jurusan Teknik Informatika Fakultas Teknologi Informasi di Institut Teknologi Sepuluh Nopember, Surabaya, Jawa Timur.

Di jurusan Teknik Informatika, penulis mengambil bidang minat Interaksi Grafis dan Seni dan memiliki kompetensi pada beberapa subjek seperti Manajemen Proyek Perangkat Lunak, Pemrograman Perangkat Bergerak, Audit Perangkat Lunak, dan Pengembangan Game. Selama berada di dunia akademi kampus, penulis aktif sebagai asisten dosen untuk mata kuliah Pemrograman Perangkat Bergerak dan Pemrograman Android. Selain itu penulis juga aktif dalam bidang nonakademik. Organisasi mahasiswa yang pernah diikuti penulis adalah menjadi Staf Departemen Riset dan Teknologi Mahasiswa Teknik Computer-Informatika (HMTC) periode 2012-2013, dan menjadi panitia maupun peserta di berbagai kegiatan jurusan. Penulis juga pernah menjadi juara pada beberapa ajang perlombaan seperti, Mobile Games Developer War 5 pada tahun 2013, Indosat Wireless Innovation Competition (IWIC) 8 pada tahun 2015. Penulis dapat dihubungi melalui alamat email rohmanhakim@live.com

[Halaman ini sengaja dikosongkan]